OGC® DOCUMENT: 21-035R1

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D022

Open Geospatial Consortium

OGC TESTBED-17: MODEL-DRIVEN STANDARDS ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2022-02-10
Approval Date: 2022-03-04
Publication Date: 2022-03-31
Editor: Ronald Tse, Nick Nicholas

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD. THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Copyright notice

Note

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 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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

I ABSTRACT

This OGC Testbed 17 Engineering Report is deliverable D022 of the OGC Testbed 17 initiative performed under the OGC Innovation Program, incorporating the D022, D143 and D144 tasks that have produced Model Driven Architecture (MDA) tools.

This ER:

1. details state-of-the-art analyses of existing MDA tools with their capabilities and limits; and
2. provides clear recommendations on how model-driven design can be fully exploited in the context of rich data model and API design efforts.

II EXECUTIVE SUMMARY

The Model-Driven Architecture (MDA) approach to the generation of Model-Driven Standards (MDS) and Platform-Independent Models (PIM) has been proven to work through Testbed activities (D022, D143, D144) covered by this Engineering Report (ER).

A total of 4 prototype systems were developed under D144:

1. Converting the CityGML 3.0 conceptual model into an MDS;
2. Converting the Discrete Global Grid Systems (DGGS) conceptual model into an MDS;
3. Converting the CityGML 3.0 conceptual model to XML Schema as a PSM;
4. Converting the CityGML 3.0 conceptual model to JSON Schema as a PSM.

The 5th prototype system for “Converting the CityGML 3.0 conceptual model to Resource Description Framework (RDF) as a PSM” is still in progress.

Under D143, a theoretical methodology was presented for using RDF as an alternative modelling approach instead of the Unified Modeling Language (UML).

It is determined that an MDA approach, possibly an organizational-wide one, would streamline the downstream creation of MDSes and PSMs, and greatly benefit all stakeholders of the model-driven process — authors of information models, standards, and the users of those models and standards.

Specifically, the usage of a single source of truth across these deliverables will guarantee a certain consistency in the deliverables, and also provide upstream feedback to conceptual model authors on potential impact of seemingly inconsequential changes.

The strong implication for using an MDA approach is that generally, the MDA approach makes any inconsistencies, omissions, or under-specifications in underlying information models much more visible, since they impact the MDA workflow directly.

In particular, the following criteria needs to be met for an MDA approach to be effective (this is duplicated in Clause 4):

1. The authoritative information model must be fully and accurately specified with regards to details that appear in the output model-driven standard.
2. The authoritative information model must be exportable to a format useable by the MDA tools. For instance, XMI output from most UML tools utilize a proprietary structure and require MDA tools to provide vendor-specific support.
3. Supplementary information to the information model, such as additional information models, guidance information for MDS output, or platform-specific configuration for PSM output, must be available in a form useable by the MDA tools.

This report recommends OGC to perform the following:

- develop and enact best practices, requirements and policies for wider adoption of MDA in OGC
- adopt the recommended open-source MDA tools to facilitate the OGC MDA approach (Metanorma and ShapeChange)
- formalize the MDA approach described in this ER to PIM-to-MDS and PIM-to-PSM generation, then making this approach available to other OGC authors
- implement documented recommendations within this ER (see at the table of contents: List of Recommendations)
- develop and deliver training on the specific topics of PIM-to-MDS and PIM-to-PSM as they are likely be used often in OGC
- facilitate development of features needed for the OGC MDA approach as described in Clause 4 through arrangements with the community, vendors or sponsors
- set standards and requirements on source information models, in order to enable the MDA approach to work effectively
- promote the adoption of cloud deployment and automated CI/CD for the PIM-to-MDS and PIM-to-PSM workflows to assist the model and standard development process
- revise and clarify the ModSpec specification, consider ModSpec as machine-readable models, creating a validation tool to validate ModSpec instances (see Clause 4.17.4)

- find ways to improve the integration between the model authoring tool and the standards authoring tools to support better cross-references (see Clause 4.17.3 and Annex C)
- investigate how OGC could benefit from existing SMART efforts such as BR|SMART in automated testing (CITE) and in the representation of standards. At least, there should be an ability for OGC standards to be incorporated into the SMART structure. (see Clause 4.17.2)
- investigate how the MDA approach can benefit to the development of OGC API and API standards
- potentially contribute the MDA approach and workflows with partner SDOs that OGC works closely with, such as ISO/TC 211

The Testbed participants would like to especially thank the project sponsors for their vision and openness, and for making this very important work available.

III KEYWORDS

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, MDA, model-driven

IV PREFACE

This ER is deliverable D022 of OGC Testbed 17.

The focus of this testbed activity was to demonstrate the business value of model-based standards, as an extension from the MDA approach for creating standards.

As literature suggests, the following are the major benefits from developing standards in a model-driven manner:

1. By maintaining standards content in the model, simplifying and decoupling the model maintenance process is possible.
2. Storing annotations and guidance about the model together with the actual model enables a single source of truth that can streamline the standards authoring process.
3. The maintenance of a single Platform Independent Model (PIM) supports automatic generation of one or multiple Platform Specific Models (PSMs), which can be directly used by developers and applications.
4. PSMs can also be treated as models in the same manner as PIMs. This means that PSMs also receive the same benefits from the model-based standards approach.

The results of the work performed in this testbed activity have confirmed that the above benefits are achievable through a model-driven approach, with the following findings:

1. A model-driven approach drives quality requirements upstream, and as such the model used as the source for a model-based standard must have dependable quality for downstream artifacts to be created accurately.
2. Rules and normalization procedures should be applied to a model being used as the basis for a model-based standard to ensure quality of generated downstream artifacts.
3. “Best practice” templates can be developed for different model types, such as for Sparx Systems Enterprise Architect Projects (EAP), Unified Modeling Language (UML) models, XML Schema Documents (XSDs), etc., enabling consistent presentation and facilitating reuse across the organization.
4. In certain conditions, when migrating standards that were fully model-based, a “hybrid” mechanism needs to be used, such as when the original standard deviates heavily from a model-driven counterpart or requires certain content hierarchies for backwards compatibility. In these cases, custom templates or templates with the ability to override “best practice” become necessary.

V SECURITY CONSIDERATIONS

No security considerations have been made for this document.

VI SUBMITTING ORGANIZATIONS

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- Ribose Limited

VII SUBMITTERS

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

NAME	ORGANIZATION	ROLE
Ronald Tse	Ribose Limited	Editor
Nick Nicholas	Ribose Limited	Editor
Chuck Heazel	Heazeltech, LLC	Contributor
Paul Birkel	Geosemantic Resources LLC	Contributor
Jeffrey Lau	Ribose Limited	Contributor
Nicholas Car	SURROUND Australia Pty Ltd	Contributor

1 SCOPE

1 SCOPE

This Engineering Report is a deliverable of the OGC Testbed 17 initiative. The objective of the ER is to demonstrate the business value of model-driven standards, as an extension of the model-driven architecture approach for creating standards.

This document describes several proofs-of-concept developed under deliverables D143 and D144 to demonstrate the authoring of model-based standards.

2

TERMS, DEFINITIONS AND ABBREVIATED TERMS

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS

This document uses the terms defined in OGC Policy Directive 49, which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

For the purposes of this document, the following additional terms and definitions apply.

2.1. Terms and definitions

2.1.1. application schema

conceptual schema (Clause 2.1.3) for data required by one or more applications

[**SOURCE:** ISO 19101-1:2014, Clause 4.1.2]

2.1.2. conceptual model

CM ADMITTED

model that defines concepts of a universe of discourse

[**SOURCE:** ISO 19101-1:2014, Clause 4.1.5]

2.1.3. conceptual schema

formal description of a *conceptual model* (Clause 2.1.2)

[**SOURCE:** ISO 19101-1:2014, Clause 4.1.6]

2.1.4. model-driven standard
MDS ADMITTED

standard created using a *model-driven architecture* (Clause 2.1.5)

2.1.5. model-driven architecture
MDA ADMITTED

software design approach for development of software systems centered around data models

[**SOURCE:** OMG UML 2.5]

2.1.6. model authoring tool

software used for authoring a *conceptual model* (Clause 2.1.2)

2.1.7. platform-independent model
PIM ADMITTED

conceptual model (Clause 2.1.2) that does not contain platform-specific concerns

2.1.8. platform-specific model

PSM ADMITTED

data model that contains platform-specific concerns

2.1.9. model transformation

model conversion from one form to another which may not preserve all semantics

2.1.10. model conversion

process that converts a data model in one format into another format that preserves all model semantics

2.1.11. stereotype

extension of an existing UML metaclass that enables the use of platform or domain specific terminology or notation in place of, or in addition to, those used for the extended metaclass

[**SOURCE:** OMG UML 2.5]

2.1.12. tagged value

attribute on a stereotype used to extend a UML model element

[**SOURCE:** OMG UML 2.5]

2.1.13. **UML profile**

predefined set of stereotypes, tagged values, constraints, and notation icons that collectively specialize and tailor UML for a specific domain or process

[**SOURCE:** ISO/IEC 19501]

2.1.14. **metaschema**

set of information models that are capable of describing other schemas

2.2. Abbreviated terms

ADE	(CityGML) Application Domain Extension
API	Application Programming Interface
DDL	(SQL) Data Definition Language
DSL	Domain Specific Language
EAP	Enterprise Architect Project
ER	Engineering Report
GFM	General Feature Model
GIS	Geographic Information System
GML	Geography Markup Language
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
MBA	Model-Based Authoring
MDA	Model-Driven Architecture
MDS	Model-Driven Standard
OCL	Object Constraint Language
OWL	Web Ontology Language

PIM	Platform Independent Model
PSM	Platform Specific Model
RDF	Resource Description Framework
SDO	Standards Development Organization
SKOS	Simple Knowledge Organization System
SQL	Structured Query Language
UML	Unified Modeling Language
XSD	XML Schema

3

INTRODUCTION

3 INTRODUCTION

3.1. Problem statement

In recent years the OGC has seen the emergence of new encoding formats, data models, and service architectures. The result has been a proliferation of standards which should form a coherent body of work. The tools required to achieve and guarantee this coherence have been lacking.

This ER, reporting on tasks D022, D143 and D144, investigates the application of MDA tools and techniques to manage the development and maintenance of a portfolio of related standards.

3.2. Model-driven architecture and its potential benefits

The model-driven architecture (MDA) paradigm (introduced by the Object Management Group (OMG) in 2001) is a design approach for the development of software systems.

One of the most important goals of MDA is to maintain a single source of truth for a system (a standard in this context). In an MDA approach, there is a single standardized definition (*truth*) from which all other derived standards and specifications are created (*derived truth*), potentially with additional contributions (*augmented truth*) by secondary sources of information (*supplementary truth*) (Figure 1). This approach promises technical interoperability of a synchronized specification stack between information systems.

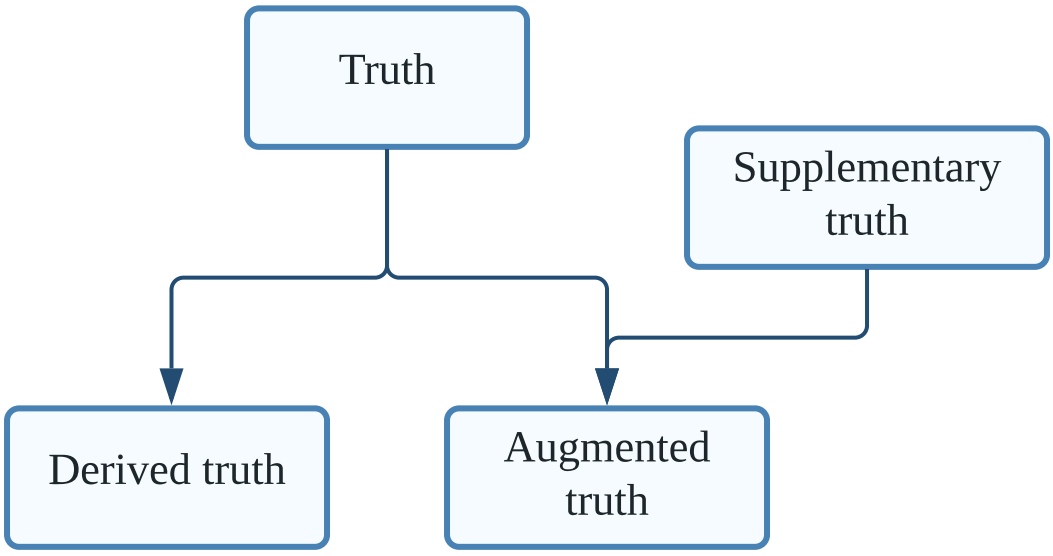


Figure 1 — Conceptual derivation of direct and augmented truths from a single source of truth

3.3. Types of models and their derivatives

Platform Independent Models (PIM) provide the definition of a data and/or computing capability which is independent of the implementing technology.

Example 1: A Conceptual Model can be a PIM, being technology-independent by definition. Conceptual Models are commonly specified using UML Class diagrams.

Platform-Specific Models (PSMs) define how the PIM capability is realized using a specific implementing technology (e.g., an XML Schema (XSD) or a relational database schema (SQL DDL)). Common implementing technologies include: XML Schema, JSON Schema, SQL DDL, and RDF.

The PSM may be derived from a PIM through a transformation taking only the PIM as input, with a deterministic mapping from PIM constructs to PSM constructs. In that case, the PSM can be spoken of as pure. The transformation may be enhanced with mapping guidance specific to the target platform (supplementary truth), in which case the PSM can be spoken of as mixed (Figure 2).

Example 2: A pure mapping from a UML Class diagram to XSD will map all UML class attributes consistently, e.g. as XML elements. If the mapping is instead to choose between XML elements and XML attributes for its rendering of UML class attributes, following user configuration, the PSM is mixed: the guidance as to which class attributes to render as XML elements and which as XML attributes is supplementary truth, as it is information extraneous to the PIM, the UML Class model.

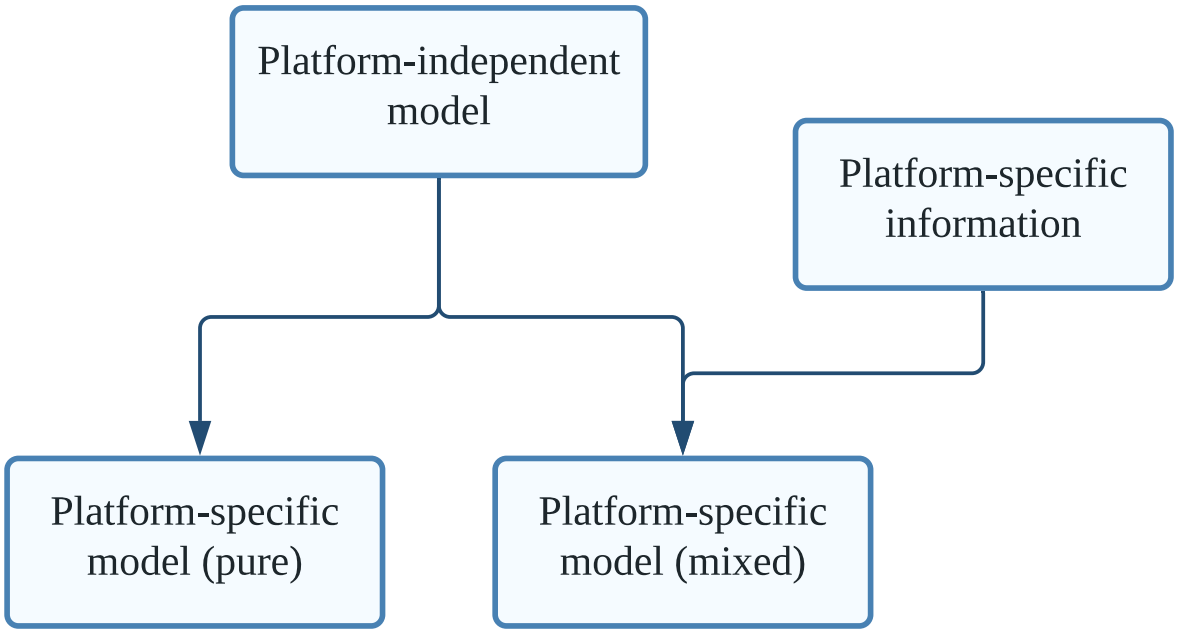


Figure 2 — Conceptual derivation of PSMs from a PIM with supplementary platform-specific information

The PIM can serve as the source of truth for the documentation describing it, as well as for downstream PSMs. Again, the documentation may be derived truth, expressing only information inherent in the PIM; or it may be augmented truth, incorporating supplementary truth outside the PIM proper — such as additional documentation or guidance (Figure 3). The latter is expected for documentation, since documentation has a tutorial and informative function, and is not just a normative description of the model. If the documentation is expressed as a standard, supplementary truth such as term definitions and bibliographic information is required in the document format. Documentation expressed as a standard is expected to have normative effect; the pure model derived from a PIM or PSM may not contain enough guidance or clarifications to have normative effect, and may accordingly be treated only as informative (Figure 4).

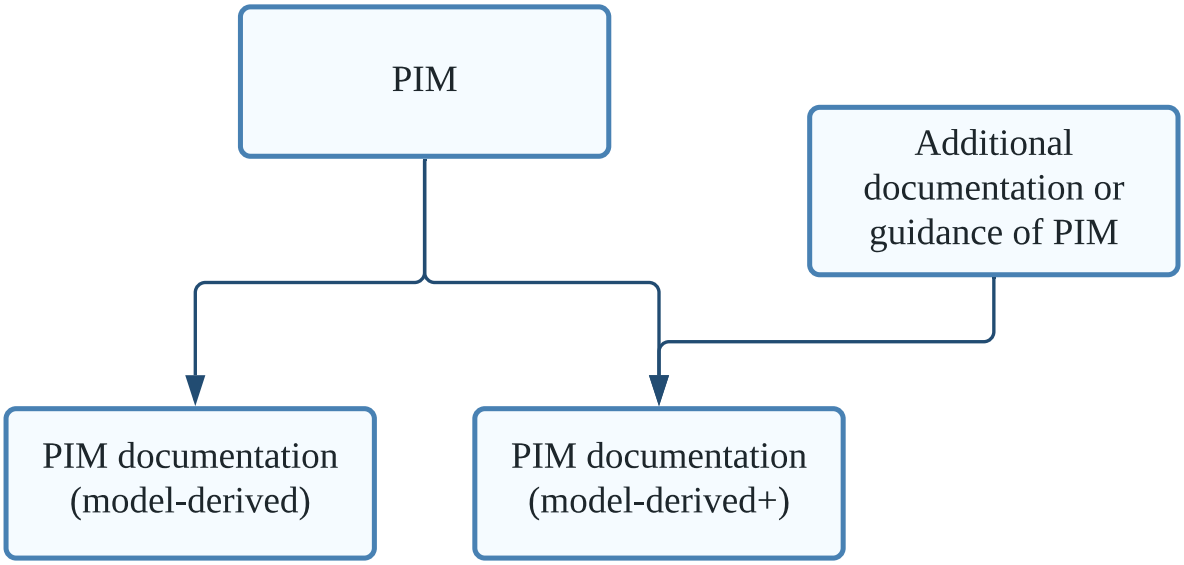


Figure 3 — Process in creating PSMs from a PIM with PSM-specific information

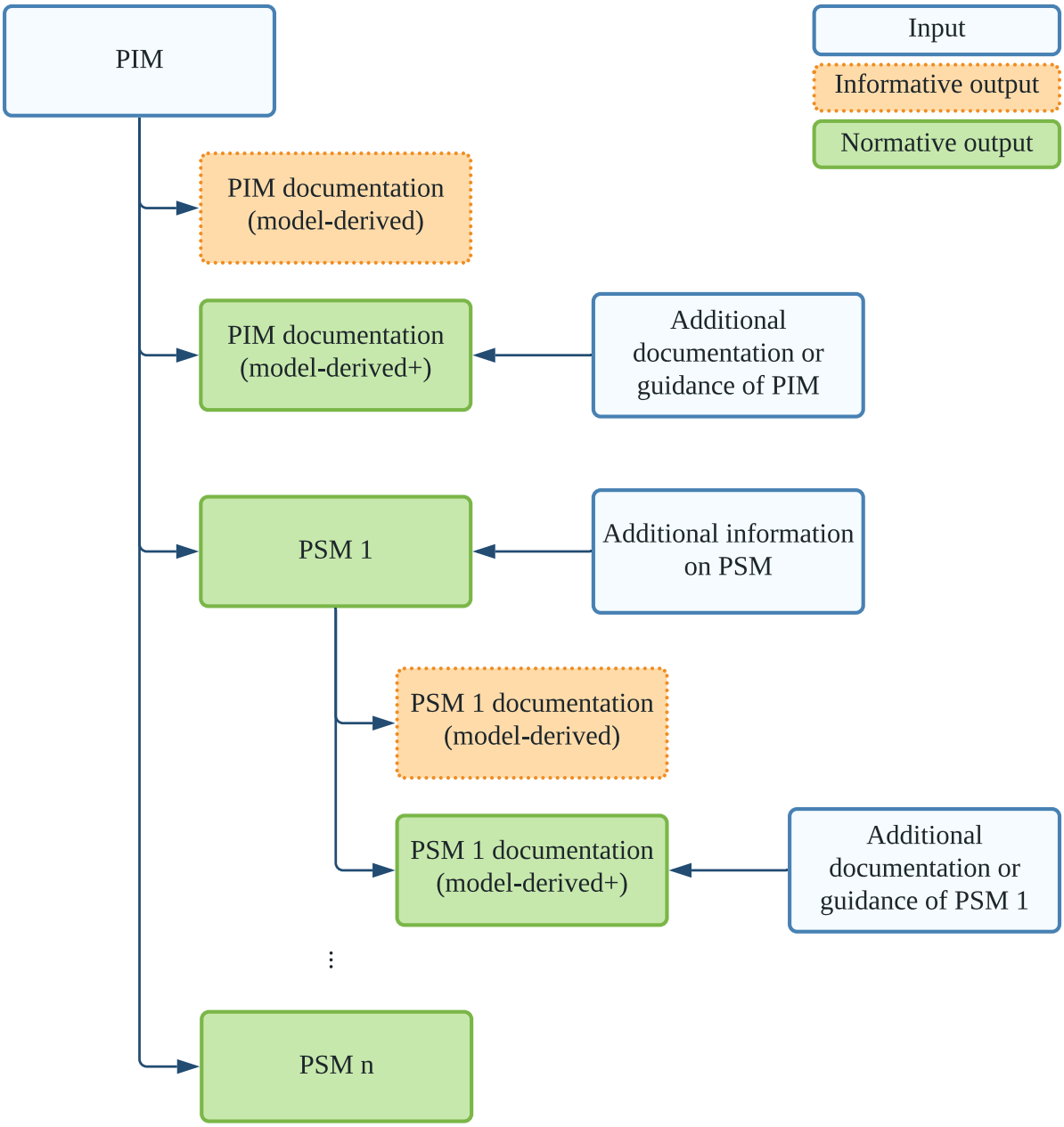


Figure 4 — PIM-derived artifacts

A derived truth (e.g. a PSM) can also be used as a source of truth to generate further derived truths (e.g. a more specific PSM). Therefore the "platform" named here can be a floating frame of reference.

Example 3: An API model (PSM) generated from a conceptual model (PIM) can further generate XML encodings (PSM) using the API model as its own starting source of truth.

3.4. MDA as applied in OGC

The OGC Innovation Program has employed model transformations in various guises as far back as the Critical Infrastructure Protection Initiative (CIPI) in 2002. In particular, the open-source tool ShapeChange has been employed in many OGC Testbed activities starting with Testbed-2 (OGC 04-100).

In the OGC, MDA approach, two types of models are defined to distinguish the cases of a source of truth against derived truths: PIM and PSM.

In OGC PIMs are referred to as “Conceptual Models”. In ISO/TC 211 PIMs are referred to as “Conceptual Schemas”. In this Engineering Report (ER) we follow the OGC terminology except in contexts in which ISO 19100-series content is being referenced.

In OGC, PSM-based standards are often referred to as “Implementation Specifications”.

Table 1 — Mapping of terminology for MDA and formats typically used in OGC contexts

MDA	OGC AND ISO	MODEL FORMATS
Platform Independent Model (PIM)	Conceptual Model/Schema	UML
Platform-Specific Model (PSM)	Implementation Specification	XML Schema, JSON Schema, SQL DDL, OWL/RDF

3.5. Challenges in developing OGC standards

While OGC has long utilized MDA for standards development, there is a recognition that gaps exist between the development of data models and the publication of those data models.

Challenges encountered include the following:

1. PIM documentation. A PIM model is often an abstract representation of requirements which require extensive documentation and guidance on the meaning of various components. Such documentation text is often managed outside of the model itself (as supplementary truth), due to tool limitations.

   NOTE 1: PIM documentation is typically published in OGC Conceptual Model standards.

2. Generation of the PSM. The PIM and PSMs may be managed by separate or independent groups. This synchronicity issue between the PIM and its derived PSMs negatively affects interoperability between information systems.

   NOTE 2: PSMs are published as artifacts of OGC Implementation Specifications.

   Example : An XML encoding (PSM) generated from version 1.0 of the conceptual model (PIM) will be out-of-sync when the conceptual model has advanced to version 1.1.

3. PSM documentation and guidance. The documentation of PIM and that of its PSMs are often managed separately. This leads to misaligned documentation and practices between the PIM and PSMs. The issue is further exacerbated as PSM documentation is meant to contain additional details specific for a platform, which by definition cannot be derived from the PIM directly.

   NOTE 3: PSM documentation is typically published in OGC Implementation Specifications.

4. Different types of deliverable outputs. For example, documentation is meant for human consumption but XML schemas are meant for machine consumption. Yet both of these deliverables may originate wholly or in part from the same model.

This ER reports on the success of addressing the first two challenges, and partially on the third.

These challenges are typically dealt with using one or more of the following strategies:

- Role segregation. For instance, adding control bodies to control and synchronize the flow of information from the PIM to downstream deliverables.
- Additional manual resources. Apply additional manual steps and corresponding resources in order to synchronize deliverables, such as in the generation of model-derived artifacts, synchronization of texts, version checking.

These challenges indicate that the following questions need to be resolved when developing a standard:

1. What is my model specification technology? Is more than one technology being used?
2. Where are the models and their documentation kept?
3. What model artifacts need to be generated?
4. What compromises can be made with regard to adhering to the MDA paradigm?

3.6. Objective: deterministic generation of PIM-derived artifacts

In this ER we report on the following objectives:

1. Identify and evaluate MDA tools potentially suitable for the generation of PIM-derived artifacts with geospatial standards, and document any gaps not satisfied by these tools.

2. Document capabilities and limits of each identified tool and provide recommendations on the context of OGC usage (don't use, adopt or extend).

    - Develop prototypes which explore solutions to identified gaps.

3. Document best practices for representing an OGC PIM from experience gained from development of the prototypes. These best practices will provide guidance to the developers of OGC Conceptual Model Standards so that those models can be used to generate valid OGC Implementation Standards using MDA tools.

3.7. Applying MDA in the authoring of OGC Standards

OGC currently publishes several types of standards that primarily involve information models, including:

- conceptual models (OGC conceptual model standards)

  Example 1: A standard with conceptual models in the UML (XMI) format.

- encodings (OGC encoding and implementation standards)

  Example 2: An encoding standard with XML schemas in XSD format, APIs in OpenAPI YAML format.

- ModSpec models (all OGC Standards)

  Example 3: All OGC Standards utilize ModSpec models for the representation of requirements and conformance tests.

An OGC Standard can be complex in structure when describing complex domains. Inherently, an OGC Standard is composed of potentially 11 different types of models, which vary as to whether they are intended primarily for machine consumption, human consumption, or both (Figure 5).

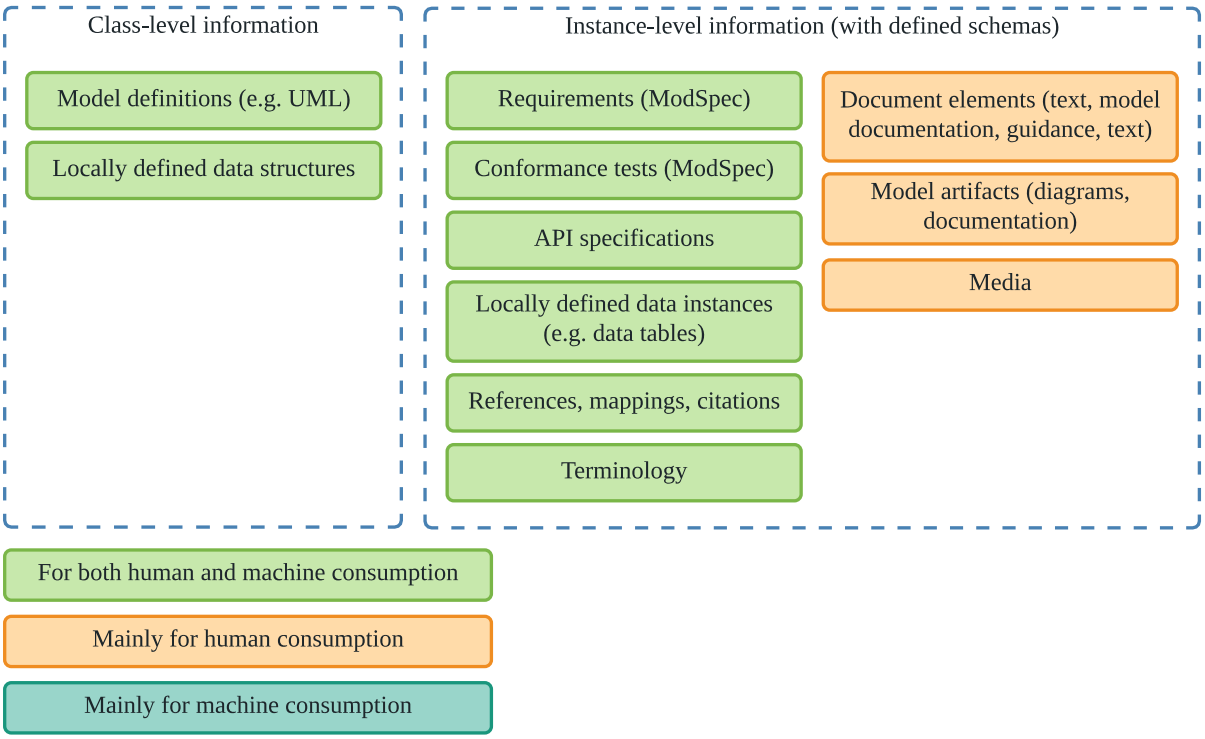


Figure 5 — Components of an OGC Standard

In order to best facilitate the creation and development of such a complex deliverable, the application of an automated, MDA-backed, generation process is desirable.

Although MDA is typically performed for software systems for the transformation of models, such as from an abstract model to an implementation model, for the testbed activity the MDA approach has been applied further to the development and publication of OGC Standards.

The application of MDA to authoring is described as “model-based authoring” (MBA), which produces a “model-driven standard” (MDS).

There are additional benefits that can be achieved:

- Providing different content to different audiences of a standard, such as:
  - Rendering human-readable standard content intended for human consumption.
  - Producing machine-readable data intended for machine reusability.
- Integration of model documentation, e.g. integrating model annotations and separate associated documentation into a single deliverable.
- Deterministic automated processing of the generation of PIM-downstream deliverables, including PIM standard documentation, PSMs and their documentation, as well as other machine-readable resources.

3.8. Achieving model-based authoring and model-based standards

Model-based standards are not a new concept. Previous OGC Standards, such as CityGML 2.0 (OGC 12-019) can be considered model-based:

- The standard contains the CityGML 2.0 model
- The standard contains content derived from the CityGML 2.0 model

The process, however, involved multiple manual steps (Figure 6), switching between tools and copy-pasting, which makes it inefficient and labor-intensive. Given that the standards development process is iterative in nature, the effort needed to keep information synchronized to publish the standard is often overwhelming.

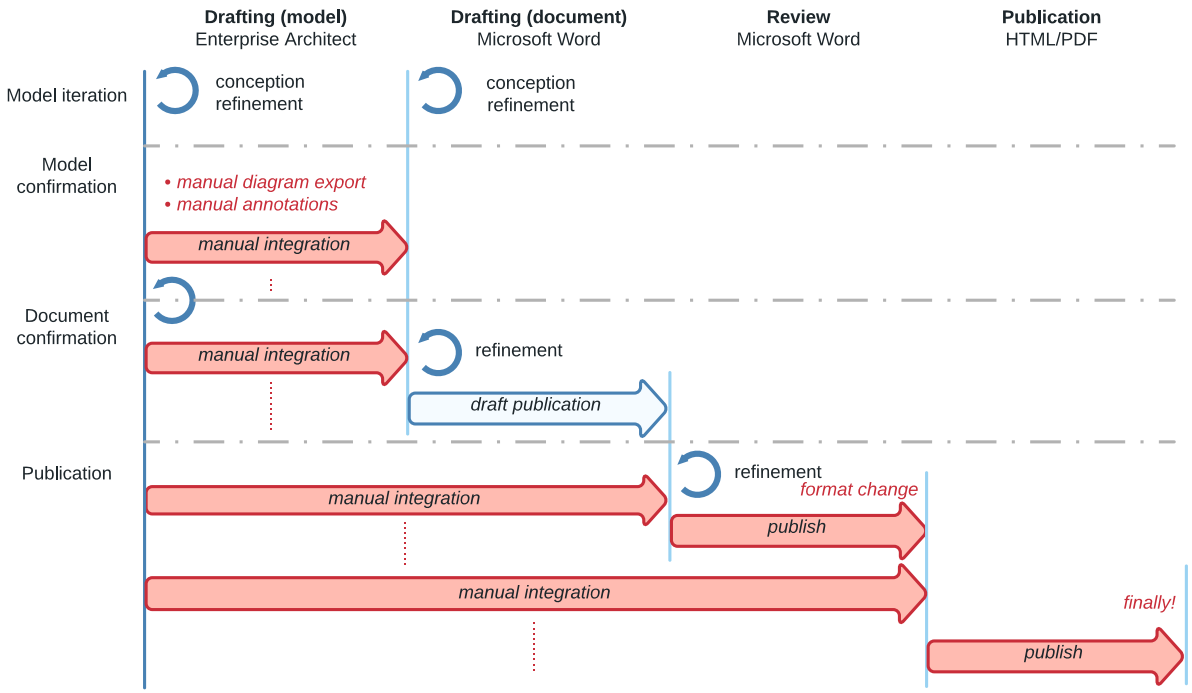


Figure 6 — Authoring model-based standards, the old way

One of the great things about model-based authoring is the potential to fully automate the process, without needing to resort to iterative and error-prone copy-pasting (Figure 7).

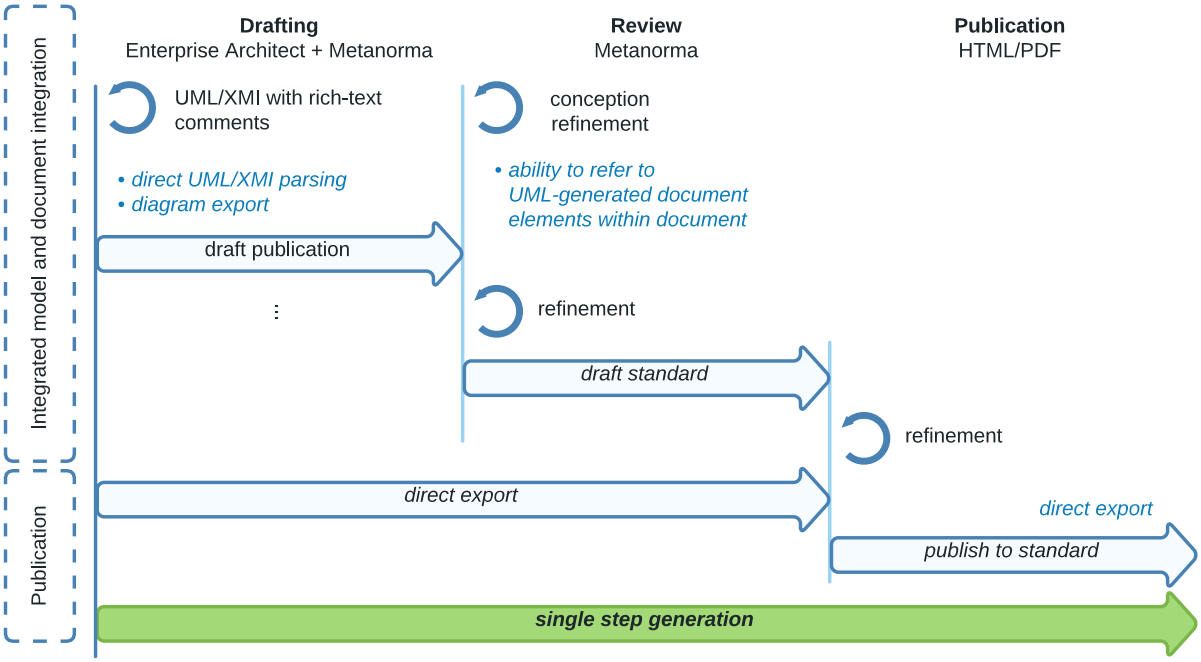


Figure 7 — New generation flow using automated processes

In order to create a model-based standard in an automated fashion, we recognize that the only way to do so is to treat all components of the standard as models.

As it turns out, text within a standard is inherently organized according to one or more information models. There exists various representations of rich-text models, including Microsoft OOXML (ISO/IEC 29500 (all parts)), OpenOffice OpenDocument (ISO/IEC 26300), as well as the Metanorma StanDoc model (ISO/AWI 36100).

Given that model transformations can be specified as defined, deterministic processes, a process can be compiled to generate a model-based standard with full automation, transforming the following inputs to outputs:

- Inputs:
  - Models to be specified
  - Text as models
- Output:
  - A model-based standard

If we treat all information that is contained in a standard as models, all that is necessary to automate the model-based authoring is to determine the model transformations from the input into a transformed set of models, and then include those transformed models into the standard (Figure 8).

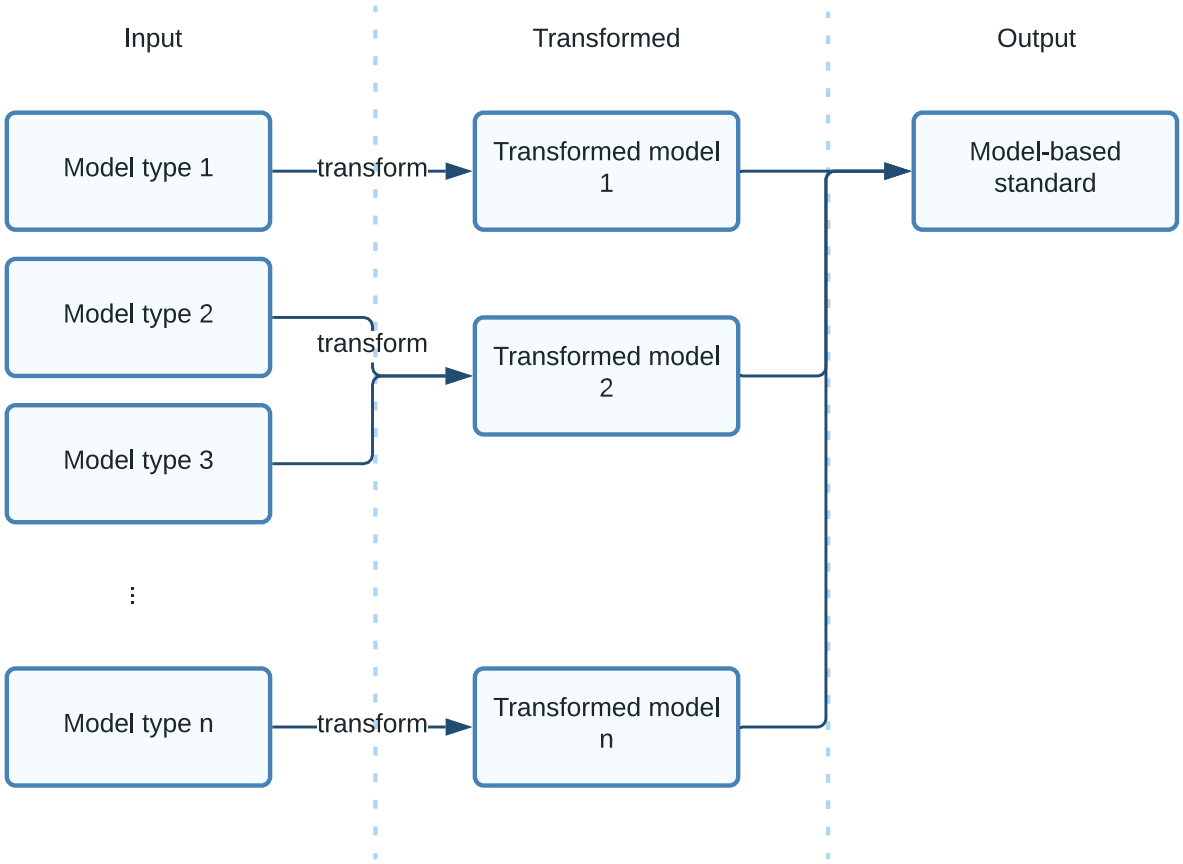


Figure 8 — Generating a model-based standard from a series of model transformations

Ultimately, there are multiple conceptual levels of models (Figure 9), expressing different levels of abstraction and interrelation between entities, and any of these can be incorporated or used inside a model-based standard.

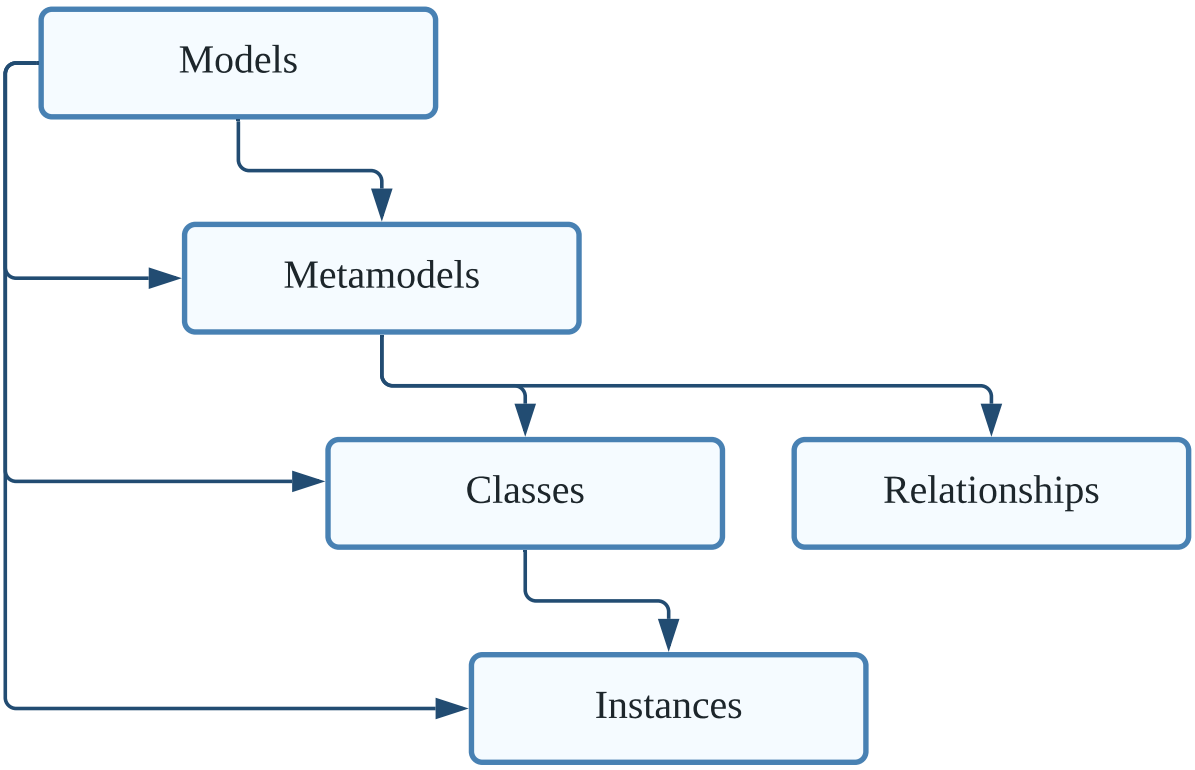


Figure 9 — Conceptual levels of information models

3.9. Benefits of automated generation of PIM downstream artifacts

There are multiple benefits by having a single-step workflow that automatically generates PIM downstream artifacts.

- Enables modern software development practices, such as:
  - Integration with change management systems (like Git) that allow distributed development
  - Usage of automated tests that ensure correctness of downstream artifacts
- Facilitates modern DevOps practices, including:
  - ability to automatically generate and distribute PIM downstream artifacts on individual changes, which can be combined with tests to ensure correctness of generated output

- Provides confidence in the correctness of PIM-derived artifacts due to them being synchronized with the input PIM.
- Reduction in the human effort necessary in validating the correctness of PIM downstream artifacts.

Ultimately, this approach "lets the computer do what a computer does best" — by elevating human effort to validate the generation process of PIM-derived artifacts instead of the validation of the artifacts themselves.

3.10. Previous Work

3.10.1. ISO/TC 211 Standards

Technical Committee 211 of the International Organization for Standardization (ISO/TC 211) manages a set of ISO standards for geographic information.

3.10.2. ISO 19109

ISO 19109:2015 defines a Unified Modeling Language (UML) metaschema for feature-based data. That General Feature Model (GFM) serves as the key underpinning for a family of ISO standards employed by members of the OGC community.

These standards define the structure and content of geospatial data in a family of "conceptual schemas" that employ the UML Class diagram methodology. These standards are regularly employed by members of the OGC community.

3.10.3. ISO 19136

ISO 19136:2007 Annex E "UML-to-GML application schema encoding rules" defines a set of encoding rules that may be used to transform a GFM-conformant UML conceptual schema to a corresponding XML Schema (XSD) based on the structure and requirements of the Geography Markup Language (GML).

The resulting XSD file(s) may be used to define the structure of, and validate the content of, XML instance documents used in geospatial data exchange.

This methodology for deriving technology-specific encodings of GFM-conformant UML conceptual schemas, based on formalized sets of encoding rules, has been successfully applied in other contexts than GML (e.g., technologies based on Resource Description Framework (RDF) encodings), and has come to be known generically as "UGAS". UGAS is an example of the Model

Driven Architecture (MDA) paradigm introduced by the Object Management Group (OMG) in 2001.

3.10.4. ShapeChange

ShapeChange is an open-source implementation of the UGAS methodology that has emerged as a key technological underpinning in many environments where platform-specific data exchange requirements have been separated from platform-independent domain (content) modeling requirements.

MDA techniques have been used sporadically to develop OGC and ISO/TC 211 standards; ShapeChange has been instrumental in many of those efforts, although effort-specific documentation has been generally sparse.

Engineering Reports documenting OGC-related ShapeChange initiatives can be found at https://shapechange.net/about/background-documents/

3.10.5. Metanorma

Metanorma is the leading open-source standards development and publishing suite.

Metanorma enables creation of semantic-aware standard documents in an end-to-end, author-to-publish workflow, based on the notion that standard documents from different standards-setting bodies rely on common requirements. It was named “Best New Semantic Technology Platform, Tools and Applications” and “Best New Scholarly Publishing Information Solution” by the International Business Awards.

Metanorma technology consists of three parts:

- a unified document model for standard documents, based on ISO/AWI 36100,
- a serialization format for standard documents into semantic and presentational formats, and
- a toolchain for authors to directly publish standard documents in the correct structure and appearance without the need for post-processing.

Metanorma is used by over 15 standards organizations, including OGC itself.

3.11. Developed PIM-derivation methodology and prototypes

In this work item the following PIM-to-PSM transformation prototypes were developed using MDA and MBA technologies.

In particular, the D144 task (Clause 10) explored the space of MDA PIM-to-PSM transformations based on two recent OGC Standards that were accompanied by formal UML-based conceptual models:

- OGC 20-010 (dated 2021-09-13)
- ISO 19170-1:2021 (dated 2021-05-11)

Table 2 — D144 PIM-to-PSM transformation prototypes and links

PIM	PSM
CityGML 3.0 CM	OGC Standard
CityGML 3.0 CM	XML schema
CityGML 3.0 CM	JSON schema
CityGML 3.0 CM	OWL/RDF
DGGS CM	OGC and ISO standard

Four target PSM environments were explored:

- Metanorma AsciiDoc (Metanorma)
- XML Schema (W3C TR xmlschema-1)
- JSON Schema (JSON Schema)
- Resource Description Framework (W3C RDF) extended by:
  - Web Ontology Language (W3C OWL), and
  - Simple Knowledge Organization System (SKOS: W3C TR skos-reference)

3.12. Semantic Web PIM-derivation methodology

The D143 task (Clause 9) explored using Semantic Web ontologies, rather than UML, as the underlying model for a PIM (W3C RDF, W3C OWL, W3C TR skos-reference). It explored how extant ontologies and vocabularies, including document ontologies, could be used to generate an MDS at the PIM level. The task did not provide an implemented prototype.

4

KEY FINDINGS

4 KEY FINDINGS

4.1. General

This section provides key findings from the prototypes developed.

It is determined that an MDA approach, possibly an organizational-wide one, would streamline the downstream creation of MDSes and PSMs, and greatly benefit all stakeholders of the model-driven process — authors of information models, standards, and the users of those models and standards.

Specifically, the usage of a single source of truth across these deliverables will guarantee a certain consistency in the deliverables, and also provide upstream feedback to conceptual model authors on potential impact of seemingly inconsequential changes.

In general, the MDA approach makes any inconsistencies, omissions, or under-specifications in underlying information models much more visible, since they impact the MDA workflow directly.

This report recommends the enactment of best practices, requirements and policies for wider adoption of MDA in OGC.

As the task of generating Model-Driven Standards (MDS) covering both PIM and PSMs is extensive by itself, the reporting for derivation of MDS from each of PIM in scope for PSMs in OGC are reported on separately.

This report is specific to considerations around PIM-driven MDS and PIM-driven PSM, where a UML model is used as the PIM.

The use of each of XML Schemas, JSON Schemas, and RDF as PSMs for PSM-Driven Standards is considered in separate forthcoming reports (Figure 10).

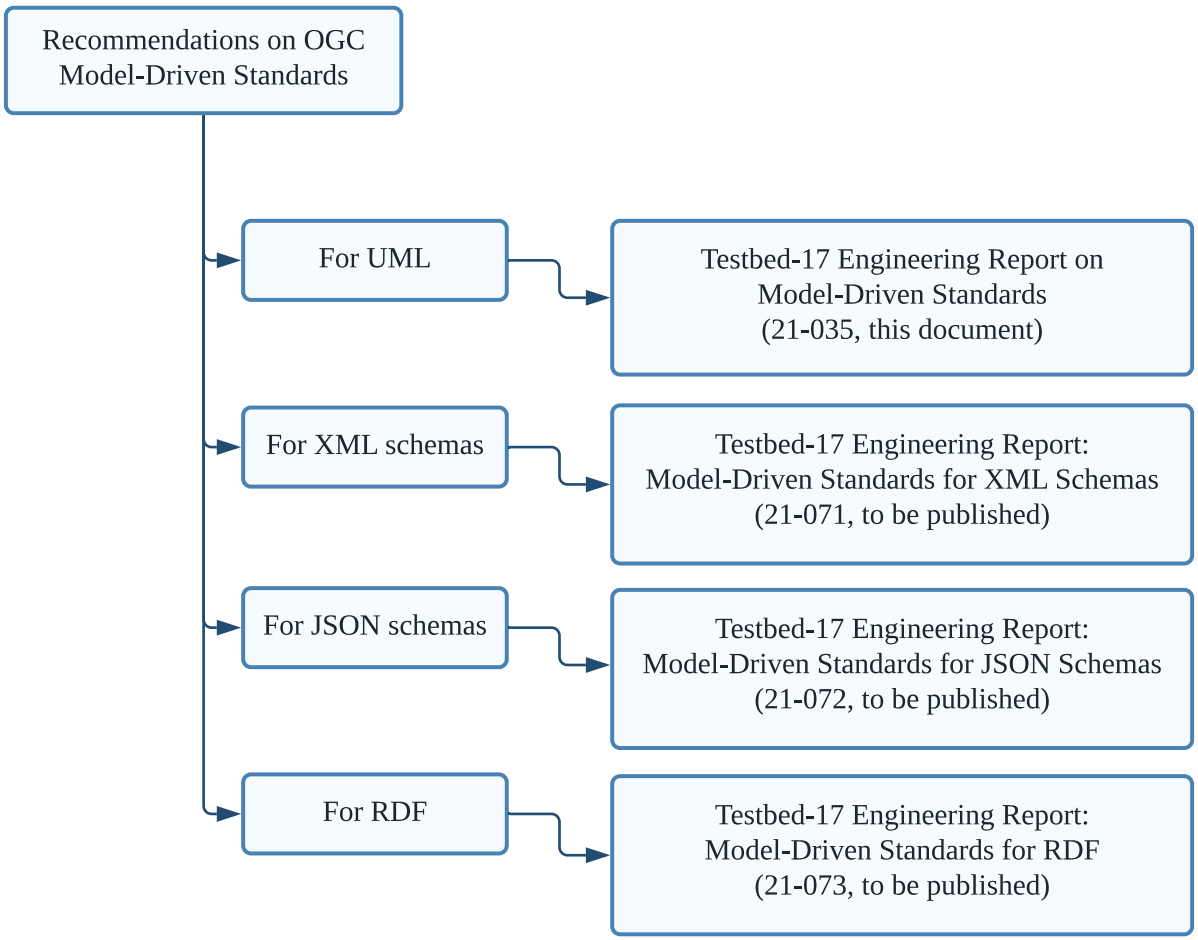


Figure 10 — Current and future OGC Engineering Reports for PSM documentation standards

4.2. Recommended MDA tools

Metanorma and ShapeChange were determined to be the best readily-available open-source tools to address the task objectives of generating MDA documents and PSMs, respectively. When used together, they form a full MDA toolchain for incorporating UML (from Sparx Systems Enterprise Architect) for OGC deliverables.

Metanorma has been demonstrated to support AsciiDoc as the input documentation format and specification of certain information models such as ModSpec. It has been successfully integrated into OGC workflows and is also used by ISO/TC 211 standards developers for creating model-based standards. Metanorma depends on Ribose LutaML, a data model accessor that supports multiple data model types, to navigate data models within the Metanorma structure.

ShapeChange has been employed by OGC and ISO/TC 211 standards developers. It has been repeatedly demonstrated to successfully employ ISO/TC 211 Harmonized Model Maintenance

Group (HMMG) conceptual schemas as input. ShapeChange support generation of PSMs as XML Schema, JSON Schema, and RDF outputs (among others).

4.3. Prototype capabilities

4.3.1. PIM-to-MDS prototypes

In the case of the PIM-to-MDS prototypes, both the PIMs (the UML classes, as expressed in XMI) and the document authoring environment (Metanorma) were stable at the time of implementation.

Metanorma is used to manage and perform the core PIM-to-MDS transformation. Metanorma is the established authoring toolchain tailored for creating OGC Standards.

Several types of information were involved in the PIM-to-MDS transformations:

- PIM and the annotations stored within it, managed in Sparx Systems Enterprise Architect
- ModSpec models, managed in Metanorma
- Supplementary text and additional diagrams, managed in Metanorma

The PIMs are exported as EA-proprietary XMI v2.5 files and EMF diagrams, as input for the Metanorma transformation process.

LutaML (Clause 7.2) and the Metanorma-LutaML plugin are used to mediate between the model authoring tool (EA) and the MDS tool (Metanorma). LutaML is a universal data accessor implemented to facilitate MDA, and has an extensible structure that allows it to read in models expressed in a range of formats, including EA-proprietary XMI.

The prototypes (Clause 10.4, Clause 10.5) have been proven successful, and the approach has been flexible enough to be useful for MDS from a range of models, not limited to the OGC, as demonstrated by the ability to generate the ISO-version of DGGS, ISO 19170.

4.3.2. PIM-to-PSM prototypes

In the case of the PIM-to-PSM prototypes, while the selected PIM (CityGML 3.0 conceptual model) was stable, the PSMs of it were not set in stone.

Given that the project sponsor encouraged experimentation, the development of the prototypes demonstrated the possibility and highlighted issues that should be addressed in order to make PIM-generated PSMs useful.

ShapeChange was used to perform the transformation from PIM to PSM. The PIM was authored in Sparx Systems Enterprise Architect.

The PIM was exported into the EA-proprietary profile of the XMI format, then processed by ShapeChange, as documented in Clause 7.1. ShapeChange is long-established as a PSM generation tool for OGC, and it supports the requisite range of capabilities to deliver the PSMs needed by OGC.

The prototypes (Clause 10.6, Clause 10.7, Clause 10.8) have been shown to be successful and demonstrates that it is indeed possible to generate a PSM that is succinct and specified to a sufficient level for application purposes.

4.4. Criteria for adopting MDA

The following criteria are deemed necessary in using a model-driven architecture:

1. The authoritative information model must be fully and accurately specified with regard to details that appear in the output model-driven standard.
2. The authoritative information model must be exportable to a format useable by the MDA tools. For instance, XMI output from most UML tools utilizes a proprietary structure and requires MDA tools to provide vendor-specific support.
3. Supplementary information to the information model, such as additional information models, guidance information for MDS output, or platform-specific configuration for PSM output, must be available in a form useable by the MDA tools.

A model-driven architecture approach to creation of MDS and PSMs should only be adopted if the above criteria are met.

4.5. Considerations for generating MDS

There are three approaches to developing model-driven standards directly from an information model.

- Automatic inclusion. This is the recommended approach that requires minimal intervention of the MDS author in the generation of the MDS from the information model. In this approach, the model content order is inherited from the model authoring tool and fully reflected as shown in the table of contents in the MDS.
- Manual inclusion. This is an advanced approach, where the MDS author wishes to manually specify content from the information model to include in the MDS.
- Hybrid inclusion. This is an advanced approach that blends automatic and manual inclusion. In this approach, the MDS author specifies portions to include information model content automatically, while being able to manually insert particular content within

the automatically generated MDS hierarchy. This approach also allows for tailoring of the order of content as generated in the MDS.

The choice of approaches depends on the difference in organization between the information model and the desired MDS structure.

In this Testbed task:

- The CityGML MDS prototype (Clause 10.4) generally adopts the “automatic inclusion” approach
- The DGGS MDS prototype (Clause 10.5) adopts the “hybrid inclusion” approach.

4.6. Cross-referencing functionality needs in PIM-MDS interactions

4.6.1. General

There are additional needs for cross-referencing when performing a PIM-to-MDS conversion, categorized in four (4) types: A, B, C and D shown in Figure 11.

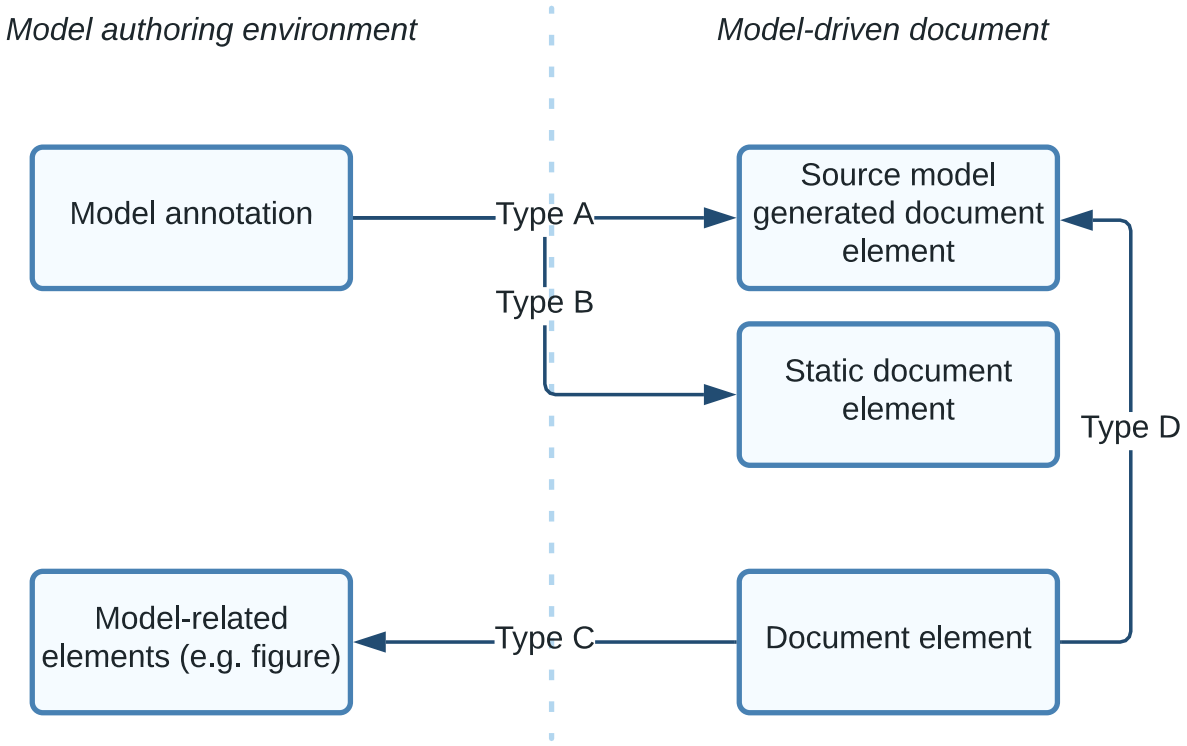


Figure 11 — Cross-references supported between PIM-MDS

4.6.2. Type A: Cross-reference model-generated document elements from the information model

This type cross-references document elements generated from the information model, from textual elements within the information model. The intricacies of this mechanism are shown in Figure 12.

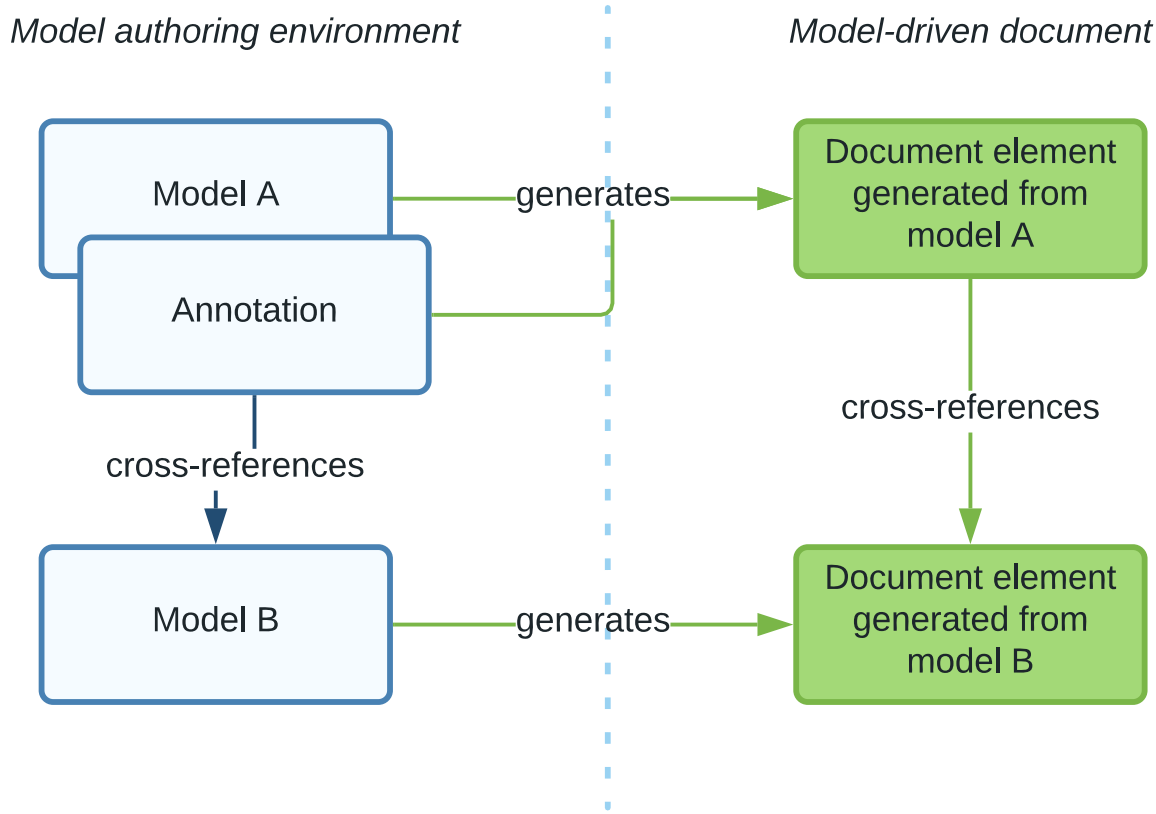


Figure 12 — Converting information model cross-references into document cross-references

In the model authoring environment, there are often textual elements of a UML model (such as the description field) that cross-reference another UML model in the same information model context (e.g. an Sparx Systems Enterprise Architect file). However, model authoring tools do not typically provide a formal and structured manner for specifying these cross-references.

The MDS tool must be able to recognize these internal cross-references through a standardized convention.

Example : In the OGC CityGML 3.0 conceptual model, there are cross-references in the description of UML classes and attributes that point to other UML classes or attributes inside the same context.

4.6.3. Type B: Cross-reference document elements from the information model

This type represents the ability to cross-reference static document elements from textual elements from within the information model.

In the model authoring environment, there are textual elements of a UML model that contain cross-references to document elements external to the information model, such as to ModSpec requirements. The MDS tool must provide a mechanism for defining referential identifiers and

allowing cross-references that utilize identifiers to be encoded through the model authoring tool, and ultimately creating the MDS with those cross-references realized.

4.6.4. Type C: Cross-reference model-imported elements from the document

In an MDS authoring environment, an expert MDS author may want to cross-reference elements that were made available through the information-model-to-MDS transformation process.

Example : In the MDS portions of the OGC DGGS standard, there are cross-references to particular UML diagrams (through the `lutaml_figure` command in the MDS) that were provided from within the Sparx Systems Enterprise Architect file.

4.6.5. Type D: Cross-reference model-generated document elements from the document

In the MDS authoring environment, an expert MDS author may want to cross-reference elements that were generated by the information-model-to-MDS transformation process.

Example : In the MDS portions of the OGC CityGML 3.0 standard, there are cross-references to particular UML package and class descriptions (through the `lutaml_table` command in the MDS) that were document elements generated from information within the Sparx Systems Enterprise Architect file.

4.7. Basic and expert features required for MDS generation tool

An extensive set of functionality in the MDS generation tool is required to support direct generation of an MDS.

These functionalities can be categorized into “basic” and “expert” needs:

Basic features include:

- Machine-readable information models. The ability to encode machine-readable requirements, ModSpec models and other encodings.
- Ability to distinguish from information models the in-scope and out-of-scope objects. Treatment of classes described in document (in-scope) vs classes not described in document (out-of-scope) (e.g. skipping, alternative references, external references).

- Ability to transform standardized and proprietary formats generated by the model authoring tool into formats desired in the MDS deliverable output.

  Example 1: When using Sparx Systems Enterprise Architect for UML diagramming, diagrams have to be exported from Sparx Systems Enterprise Architect. However, Sparx Systems Enterprise Architect is unable to generate platform-independent vector graphics (W3C SVG), but only Microsoft EMF (EA-EMF is also incompatible with standard EMF). As OGC requires vector graphics to be rendered in platform-independent formats for HTML, PDF and Word outputs, the MDS tool must handle conversions from EMF to SVG.

Expert features include:

- Ability to configure presentation styles of classes. For users of the "automatic inclusion" approach, an OGC-predefined display template is used to transform a UML model in into textual elements (e.g., a UML class uses a display template for showing a UML class in textual elements). An expert MDS author may want to customize how a UML class is represented.

  Example 2: In the CityGML 3.0 standard, Clause 7 and Clause 8 use different presentational structures on the same set of UML models: Clause 7 uses the `data_dictionary` template, while Clause 8 uses the `entity_list` template.

- Ability to filter and order UML packages and classes. For users of the "automatic inclusion" approach, the content order of UML packages and UML classes is inherited from the input information model. An expert MDS author may want to use a different order in the MDS from that of the original information model due to other considerations.

  Example 3: In the CityGML 3.0 standard, the content order in the MDS differs from that of the CityGML conceptual model in order to match and align the clause numbers with the CityGML 2.0 standard to allow comparisons between two standards.

- Ability to insert document elements before and after each UML package / class presentation. In OGC conceptual models, ModSpec requirements and conformance tests are generally defined outside of the conceptual model itself. When using the automatic inclusion approach, an expert MDS may want to place ModSpec requirements and conformance tests right after the description of a particular package or class.

  Example 4: In the OGC DGGS standard, ModSpec requirements and conformance tests are encoded in the Metanorma ModSpec format. These ModSpec models are inserted into the "automatic inclusion" process through conditional insertion procedures provided by Metanorma's LutaML plugin.

- Ability to distinguish model elements that exist within the current PIM or are external to the current PIM. A UML conceptual model may refer to externally-defined UML packages and models that do not exist within the currently loaded package. The MDS tool must distinguish whether a referenced model contains such reference, and create that cross-reference accordingly. There are two possibilities:

  - A referenced model is not available in the current package. The cross-reference will have to be made to either the specification of the referenced model, to an alternative provided by the user, or omitted completely.

- A referenced model is within the current package but is not documented in the MDS, intentionally or not. In this case the cross-reference can only be made if the referenced model exist in the MDS, or if an alternative link was provided by the user.

Example 5: In the CityGML 3.0 conceptual model, there are multiple ISO-related UML classes referenced within the models but the classes are actually located outside of the CityGML 3.0 conceptual model. These external models instead are represented within the MDS as terminology entries in the Glossary, and hence cross-references need to be made from the PIM-generated document elements to those Glossary entries where appropriate.

- Ability to provide a dynamic context for programmatic document elements. In the MDS authoring environment, an expert MDS author may want to provide a template for generating document elements to be inserted right after the description of every UML package, but the content generated is context-dependent.

  Example 6: In the OGC CityGML 3.0 standard, every Clause 8 subsection represents a UML package. At the end of each of these subsections, a section is inserted to provide a link to that particular package's documentation in the OGC CityGML 3.0 User Guide. This link depends on the package name of the UML package. With Metanorma providing the dynamic context through the Liquid template language, it is possible to encode this insertion with a single block instead of one block per subsection.

The need for such functionalities is discussed further in Clause 7.2, Clause 10.4 and Clause 10.6.

4.8. PIM-to-PSM transformations rely on supplementary truths

With the CityGML 3.0 conceptual model, the developed prototypes documented in Clause 10.7 and Clause 10.8 demonstrated that the generation of PSMs from PIMs are both feasible and practical.

However, the following considerations must be taken into account:

- PIM-to-PSM transformations are, at their core, model conversions. Different information modelling languages present different semantics, and the source semantics may not be fully expressible in the target modelling language. On the other hand, a semantically lossless representation in the target modelling language may be too convoluted to be useable in the target use case.

  Example 1: UML is not trivially converted into W3C RDF given the different contextual assumptions of those languages. A full UML representation in RDF requires an implementation of UML metamodels in W3C RDF, of which the result would not be useable in traditional RDF scenarios.

- There are multiple methodologies for converting a PIM to PSM, and there is not necessarily an "absolutely correct" methodology. The choice of methodology will likely depend on the target usage case that dictates the usage of the target information

modelling language. An organization-wide approach, for consistency reasons, should take into account what conversion methodology should be allowed.

- Programmatic guidance for the PSM-generation process, as specific to a PSM, is necessary. These are considered “supplementary truths” necessary for creation of the PSMs. There are two places where such conversion guidance (or “hints”) can be located:

  - Within the conceptual model

    Example 2: In the DGGS conceptual model, XML schema generation hints are provided within the UML models.

  - In configuration input for the PIM-to-PSM generation tool

    Example 3: In the case of the CityGML 3.0, the XML schema generation configuration is placed external to the conceptual model as a separate ShapeChange configuration file.

The management of this PSM-specific information and configuration should be considered part of any MDA approach.

4.9. Considerations when using UML models as PIM

The goal of using MDA is to derive output without relying on additional sources if the necessary information already exists in the authoritative model.

The following findings have been made with regards to PIM-to-MDS and PIM-to-PSM transformations:

1. UML models exported into XMI from one model authoring tool are typically not interoperable with another tool unless vendor-specific support is implemented. The MDS tool must provide vendor-specific support for the model authoring tool used.

   - While the machine-readable form of UML is specified in the XMI standard (OMG XMI 2.5), the XMI profiles of nearly every UML authoring tool differs — only the most basic core UML models are interoperable. Only some UML authoring tools implement compatibility features to support the import and export of UML XMI files in order to support XMIs from different vendors.

   - Both conceptual models used in this Testbed activity, CityGML and DGGS, are authored in the Sparx Systems Enterprise Architect EAP format. A machine-readable UML export file from Sparx Systems Enterprise Architect, in form of XMI v2.5, consists of two separate XML portions. The first portion is the XMI-standard conforming portion, which allows for interoperation between tools. However, this portion omits critical information of UML models needed for documentation, such as the description of models, stereotypes, and diagrams. The second portion is proprietary XML content,

which provides most necessary information supported by the Sparx Systems Enterprise Architect GUI for the documentation of UML models.

- In the MDS prototypes, the LutaML-XMI plugin had to be extended to support Sparx Systems Enterprise Architect specific XMI parsing in order to retrieve the necessary information for the documentation of the CityGML and DGGS conceptual models.

2. The authoritative model must be **fully specified** for an MDS approach. To enable automated tools to utilize information from a model, the model must provide its information in a machine-interpretable manner and allow distinction between specific types of information. In a PIM-generated MDS or PIM, any under-specification or aberration in specification practice will cause unintended failures downstream. It is therefore important to only standardize a conceptual model after its MDS is generated.

   - To support PIM-to-PSM XML Schema transformation, the CityGML 3.0 Conceptual Model (OGC CityGML 3.0 Conceptual Model) was developed in conjunction with software modifications to a non-current ShapeChange baseline, in order to treat CityGML-specific "core" «`BasicType`» `DoubleBetween0and1List` and `DoubleList` generalization classes on a special-case basis.

     To remove this dependency, the current ShapeChange software baseline was extended with one new XML Schema-related rule (`rule-xsd-cls-basictype-list`), and the CityGML 3.0 Conceptual Model revised accordingly. This rule is independent of the CityGML 3.0 Conceptual Model, addressing the more general topic of the modeling of lists of basic types in Application Schemas for use in developing a corresponding XML Schema-based PSM.

     NOTE: If the CityGML Standards Working Group (SWG) chooses to adopt these minor enhancements to the CityGML 3.0 Conceptual Model and XSD-based encoding then CityGML Application Domain Extension (ADE) developers will be able to use the ShapeChange capability to generate their community-specific XSD-based encodings.

   - The DGGS conceptual model was used without revision for PIM-to-MDS generation.

3. Data elements within a model that have to be documented in the MDS must be specified in a standardized manner and in the correct format. There are potentially multiple ways of achieving similar results with a UML tool like Sparx Systems Enterprise Architect, especially when model authors focus on the

diagramming (drawing) functionality and are unaware of best practices on how and where to specify information.

- The DGGS conceptual model utilized practices that were problematic in information extraction due to the lack of support of those model elements in the modelling language or the model authoring tool.
    - Sparx Systems Enterprise Architect does not allow specifying “inactive” or “outdated” elements inside a an Sparx Systems Enterprise Architect project file (it does allow for specifying “draft status”). In the DGGS conceptual model, there were several types of “inactive” elements that are not supposed to be included in the MDS, including:

        A) Spare model elements that were stowed for reference purposes. These were kept in a UML package called “Spare”.

        B) Outdated model elements in various UML packages for reference purposes. These packages, classes, interfaces had names prefixed with the phrase “old:”.

        C) Non-normative diagrams for reference purposes. These diagrams are prefixed with the phrase “fyi:”.

    - OMG UML XMI does not provide a mechanism for specifying the “dimension” of a UML operation. The DGGS conceptual model relied on differentiating the “dimension values” of UML operations encoded in the “Notes” field, but this information is clearly not machine-readable. (https://github.com/metanorma/ogc-dggs-xmi/issues/21)
    - The DGGS conceptual model also depended heavily on OCL, but Sparx Systems Enterprise Architect does not support validation of OCL. In the conceptual model, there are OCL entries that used idiosyncratic identifiers and content that are invalid in OCL grammar.

        Example 1: In the DGGS conceptual model, the usage of `<<query>> (1D)`, to indicate that the data type of a relative position should be one-dimensional, conflicts with `<<query>>` that is supposed to mean a two-dimensional relative position. It also used a parenthetical qualifier to restrict the type of the UML attribute.

4. Naming of diagrams within the model authoring tool should be consistent.
    - The DGGS conceptual model utilized a prefix of “Fig:” for the names of all diagrams encoded in the UML packages. In the MDS, such prefixes have to be removed as they would pollute the actual name when represented in a figure.

5. Naming of UML packages within the information model should be easily referable and unique.

    - In the DGGS conceptual model, most UML packages utilize long names that include spaces and punctuation characters, such as “Common Spatio-Temporal Classes”. While these names prioritizes human-readablity, they present a challenge to machine-referencing.

      The useability challenge of these long names is compounded when accompanied by a hierarchical UML package structure. When using UML models with MDA, the models need to be referred to via fully qualified names given the possibility of conflicting model names across UML packages. These fully qualified references are not only cumbersome for MDS and PSM authors, but are also uncomfortably long for any human audience.

      Example 2: In the DGGS conceptual model, a unique reference to a particular UML class can look like “Common Spatio-Temporal Classes > Temporal and Zonal Geometry > Temporal Geometry and Topology > EdgeT”,

    - While model authoring tools such as Sparx Systems Enterprise Architect provides a unique reference per model element and diagram, these unique IDs are long and very difficult to reuse without automated help from the MDS tool, e.g. the CityObjectGroup diagram in the CityGML 3.0 model has the EA GUID of `{CCE69980-0DDD-4975-9104-054B018BAA96}`.

Generally, as demonstrated by the issues under Clause 8.2, it is essential that the information model being used as input for MDA rigorously follows the model specification that MDA tools expect to see.

Lax usage of UML, while may only have minimal consequences so long as the UML only output diagrams or OGC-tailored PSM conversions, will frustrate MDA tools that are aligned with standard representations of UML.

The stakes are higher in MDA, and the processes far less forgiving of eccentricities, such as non-standard multiplicity notation (Clause 8.2.3), ad hoc usage of novel stereotypes (Clause 8.2.7), under-specified modelling of schema reuse (Clause 8.2.8), and so on.

Standards and requirements on PIM information models need to be set in order to align expectations amongst the three stakeholders in MDA usage:

- model authors
- model authoring tools
- MDA tools

4.10. Level of specification of input models (PIM and others)

As information models are being called upon in MDA to generate complete documentation, any past under-specification or omission of model components for expediency will be correspondingly reflected in the output.

Example 1: In the DGGS conceptual model, some packages were found not to be present in the ISO/TC 211 Harmonized Model.

Example 2: Similar issues with incomplete conceptual model dependencies were encountered repeatedly for CityGML (Clause 8.2.4, Clause 8.2.5, Clause 8.2.6), and point to past lax management of the information model.

Various strategies can be used to remedy insufficient specification:

- Merge multiple pre-existing models together, or use those models separately, for a single MDS. The MDS could iterate through multiple source information models, if they cannot conveniently be merged into a single source of truth.
- Treat extraneous information models as external bibliographical references, without resolving them in the MDS. However, any external models in a PIM still need to be resolved when generating a PSM, as inclusions.
- Omit problematic objects from the PIM in the MDS.

4.11. Improvements required in ModSpec for specification as PIM

OGC ModSpec (Clause 5.4) provides a set of information models for defining requirements and their conformance tests, and its usage is mandatory within OGC Standards.

ModSpec was originally designed to apply to UML and UML diagrams. In this Testbed activity, it is used as a PIM specification mechanism, which is then transformed into a PSM. Specifically, a ModSpec model instance is specified in the Metanorma ModSpec language (the PIM), which is then transformed into a Metanorma Requirement model in XML (the PSM).

A number of issues were discovered in the rigorous usage of ModSpec in the PIM-to-MDS prototypes (Clause 10.4 and Clause 10.6), that require additional specification and clarification in the ModSpec specification.

These issues are documented in detail at Annex D and summarized below:

- Clarification between the meaning of Conformance Test Step and Conformance Test Part, and the relationships between them. (Annex D.1)
- Clarification and specification of Conformance Tests and Test Steps that apply under a certain condition. (Annex D.2)
- Review and update OGC-wide guidance on usage of ModSpec terminology in OGC documents. (Annex D.3)
- Review and update the OGC ModSpec specification to address inconsistencies in its UML diagrams, UML model definitions and textual descriptions. (Annex D.4)

4.12. Administrative controls needed for adopting MDA organization-wide

If the MDA approach is applied organization-wide, administrative controls should be implemented (policies and procedures) to streamline development of PIM-generated output, namely the MDS and PSMs.

Mandatory practices (best practices for MDA) should be established and documented that covers the following MDA activities:

- PIM model requirements. Specifically, requirements for a UML model being used as a PIM model. This covers the correctness, accuracy and level of specificity of the UML model when used as a PIM.
- PIM to MDS specification best practices. When using a PIM to generate MDS, it is important to have all information intended in the MDS available in the PIM. In addition, since the MDS contains detailed specifications of the PIM, the PIM must make available its information in correctly specified semantics.
- PIM to PSM specification best practices. The generation of PSMs depend on the general health of both the PIM and PSM-generation configuration or guidance. An incorrectly-specified (or under-specified) PIM or an incorrectly configured PIM-to-PSM process will not generate the desired PSM results.
- PIM to PSM to MDS best practices. This topic will be addressed in forthcoming reports (Figure 10). Simply put, the transformation to PIM to PSM to MDS involves both passing through truths generated from the PIM-to-PSM steps to the PIM-to-MDS step (where the PIM is the PSM output of the former step).

4.13. Sequence of content in MDS

In this Testbed activity, both PIMs (CityGML 3.0, DGGS) are implemented with a hierarchy of UML packages.

The information in the source information model may be presented in different sequences in the MDS according to external requirements. The sequence of content impacts how the integration of data from the model will be done, and needs to be planned for.

Take UML, as it was used in this Testbed activity, as an example. UML allows for the creation of package hierarchies, where a UML package can have sub-packages, and so on. This allows for a compartmentalized approach that promotes separation of concerns, potentially preventing confusion in its audience if the models were numerous.

In UML, UML objects within a UML package can be given a particular “sorting order”. Similarly, UML packages themselves, being UML objects, also belong to the same sorting order. The sorting order may or may not be alphabetic.

An MDS document, being a document, naturally provides a sequential sequence of content. When transforming a PIM into an MDS, the author must consider whether the UML object sorting order and the UML package hierarchy are to be reflected in the generated MDS.

It may therefore be necessary to indicate the ordering of model objects in the MDS through a declaration overriding the ordering of objects within the model; the ordering is specific to the target document, and not to the model.

NOTE: It is said that Sparx Systems Enterprise Architect supports custom ordering of UML packages. If so, it is possible to have the MDS tool (Metanorma) support that encoded ordering if desired.

The decision on whether to adopt the UML object sorting order and package hierarchy may be based on the following considerations:

- Backwards compatibility to a previous edition of the standard. If there is a previous edition of the MDS, an author may wish to retain clause numbering to match the previous edition, and have new UML elements appear after the last clause (or subclause) of the previous edition.

  Example 1: The CityGML 3.0 MDS opted to use a “hybrid inclusion” approach with a fixed content sequence in order to retain clause numbering of the previous edition (CityGML 2.0 standard). Description of new model elements were appended after the last subclauses of the previous edition.

- Conventions. In some OGC conceptual model standards, a convention is used for describing a UML package that first provides overview diagrams of the involved models,

then tables and text that describe the details of those models. Each detailed section of models starts with the corresponding ModSpec requirement instance.

Example 2: The DGGS MDS uses the "automatic inclusion" approach, which implements a default order of first showing information of the UML package, then the diagrams it contains, then UML classes, then UML relationships.

- Specific ordering needs. For a PIM based on UML, for example, classes may be presented aggregated by package, by diagram, or by model, and they may be sorted alphabetically or through dependency ordering.
- Hierarchy presentation or flattening. UML packages can be nested multiple levels, and having a deeply nested document hierarchy makes the standard hard to read, and there is a possibility of running into governance issues (ISO only allows a maximum of 5 clause levels). An author may wish to free the document clause hierarchy from the UML package hierarchy.

In the prototypes (Clause 10.4, Clause 10.5), the following approaches were used:

- In the DGGS prototype, the UML classes of the DGGS were presented in the same order as they were expressed in the source PIM, so the LutaML integration merely iterated through the UML classes in a single loop.
- In the case of CityGML, the UML classes were presented in two clauses, one providing an overview of the models (20-010, Clause 7 "CityGML UML Model"), and one providing attribute detail (20-010, Clause 8 "CityGML Data Dictionary")
  - This meant that the UML classes were iterated through in two separate loops, each extracting and presenting a different level of detail.
  - The order of presentation of the CityGML UML classes was also different from that in the PIM, with the classes already present in the previous version of CityGML presented first.

The differences in how the two documents presented their information may be summarized in Table 3.

Table 3 — Comparison of LutaML approaches between CityGML and DGGS

	CITYGML 3.0	DGGS
Document structure	Does not align with package hierarchy • Requires backwards compatibility to City GML 2.0 standard document structure • Requires manual references to generated package/class/figure document artifacts	Identical to package hierarchy • Including packages, classes, figures
Model representation structure	Flat	Hierarchical

	CITYGML 3.0	DGGS
Model rendering style	Multiple styles on the same content: entity list vs data dictionary	Single

4.14. Challenges with coordinating derived and supplementary truth

4.14.1. General

Supplementary truth applied to an MDS or a PSM is extraneous to the primary source information model, by definition, so it has to be planned as a separate activity outside the generation of the augmented truth.

4.14.2. Challenges for MDS authors

In fact, MDS authors are required to fill in the gaps in guidance that are not apparent from inspection of the model in isolation. This implies that authoring happens in passes, rather than authoring both model definitions and commentary in a single go:

- the author first generates the derived truth: an MDS presenting the PIM
- then works out what additional information needs to bind to the derived truth, and where to do so.

The MDA approach to MDS implies that authoring is a non-linear process. Authors will need to refer back to notes and comments to explain design decisions and constraints. Authors that are not familiar with the primary source information model (the PIM) may be uncomfortable with writing guidance for it.

On the other hand, given that the model is documented automatically out of machine-readable artifacts, which have typically already been used in production in an automated, well-engineered software process, means that details will not be missed, and that the completeness of documentation is guaranteed.

Authors used to working with linearly-authored documentation may resist the constraints of embedding documentation into unfamiliar model authoring tools, especially if they are not fully integrated with their authoring environment.

It may prove necessary to make future information model integration into documentation configurable, so as to skip specific annotations, which will be addressed in supplementary truth instead.

An MDS author intending to create OGC deliverables is naturally required to work natively with information models and cannot shy away from the following repertoire:

- PIM model in UML
- ModSpec models in Metanorma or YAML format
- Text markup in Metanorma format

4.14.3. Challenges with collaboration of MDS content

In the MDS prototypes for CityGML 3.0 and DGGS, the supplementary truth for the documents has been written as Metanorma text, and managed through Git in order to enable collaborative authoring and review of that text.

As with any collaboratively authored text, a certain amount of discipline is required around authoring, particularly as the structure of the supplementary truth is dependent on the given information model: the editor needs to ensure that the connection between supplementary truth and the information model is maintained.

There is an added coordination challenge where ideas need to be generalized across different component implementation teams for the same standard, working from the same information model.

4.14.4. Challenge to coordinate PIM annotations and supplementary truth to MDS

A further challenge is that some textual content of the MDS is not extraneous to the information model, but is included in the model as annotation.

The division between the two kinds of content needs to be planned for, to prevent confusion and duplication of effort, especially if the annotations and the supplementary truth are managed by separate teams, and risk either contradicting each other, or presenting information redundantly.

Coordination was proved necessary in the prototype: there needed to be guidance offered on what information to enter in Notes, and what syntax it should be expressed in (which has implications for the integration process).

It may be necessary to constrain the kind of content that can appear in either derived or supplementary truth, to prevent conflict and confusion.

In the MDS prototypes, LutaML iterates through objects from the information model as a block. So it presents each UML class derived from the PIM as a complete entity in the target Metanorma document; it would do the same for an object definition in XSD.

However, it is not currently possible for LutaML to interleave supplementary truth authored outside the model authoring tool with the presentation of an information model object: any commentary or annotation on a model object needs to be added after the object is

presented in full. So authoring an MDS is somewhat inflexible: objects from the derived truth (including annotations out of the model management system) are presented as a block, and any commentary or discussion of the object need to occur as a block after the object. There is no current mechanism, for instance, to annotate particular parts of the object with comments as callouts, written in Metanorma.

Example : A note on a UML attribute cannot be interpolated from Metanorma into the PIM-derived tabular presentation of the class: it can only be added before or after the object rendering (provided for in LutaML as `[.before]` vs `[.after]` blocks).

This limitation leads to a division of labor in the current tooling: annotations within an object have to be added in the model authoring tool, and cannot include supplementary truth.

This challenge is an instance of the question of how to annotate externally sourced online content. That has been a longstanding concern with hypertext, and approaches have been devised to tackle it (e.g. W3C TR annotation-model). If interleaving supplementary and derived truth more fluidly in MDA proves desirable, LutaML can be expanded to support it better, harnessing such approaches to annotation.

That said, whether interpolating supplementary truth into model-derived artifacts is even desirable is unclear: it makes for more flexible authoring, but it can also lead to more difficulty in coordinating authoring, and it can make the document harder to read.

4.15. Tool experience and limitations

In the development of prototypes for the PIM-to-MDS and PIM-to-PSM activities where the PIM is a UML model, multiple compatibility gaps were discovered in the usage of selected tools for the MDA approach.

While all compatibility gaps have been closed as a consequence of this Testbed activity, mainly through the efforts of the Metanorma and to some extent the ShapeChange development teams, certain gaps that arise due to limitations of tool choices remain to be resolved.

Sparx Systems Enterprise Architect while being a popular tool for authoring UML models, provides generally a large amount of freedom to the model author to create UML models that are not conducive to the machine-interpretation functionality needed for a successful MDS approach:

- EA provides a rich-text content area for annotations and notes for UML models and diagrams. However, this rich-text content area uses an outdated format (RTF) and is entirely independent from other UML models defined in the same model package. This means that content within a note in a UML model is unable to specify cross-references to another UML model in the same EA project.
- EA allows entering OCL, which is commonly used with UML models, in specifying interfaces and machine logic. EA however does not implement any validation of OCL, which means that a downstream application using EA output can encounter erroneous OCL.

- EA provides export capability for its diagrams. Typically, UML diagrams are best rendered in vector formats since they contain heavy use of shapes. However, EA can only export in the outdated, obsolete and proprietary Microsoft EMF format, which cannot be used in applications outside Microsoft Windows, as described in Clause 10.4.3. The EA implementation of EMF is even worse — it does not even conform to the Microsoft EMF specification — a correct implementation of EMF will display some objects in the EMF in a vertically flipped manner.
- EA itself is not a cross-platform application. EA only works on Microsoft Windows. Moreover, EA itself is still currently a 32-bit Windows application despite 64-bit Windows being available as mainstay for years (EA Release 16, which is 64-bit, is in Beta as of January 2022). In OGC, many users utilize macOS and Linux systems, and in order to use EA they are forced to install the free and open source Wine software product, or the commercial flavor of CrossOver (the latter requiring an annual subscription). Note that EA has been proven to work on Ubuntu 20.04 LTS through version 7.0 of Wine.
- EA itself is not suitable for usage in cloud systems, continuous integration and automatic generation of downstream model artifacts due to licensing limitations and the delivery model as a Windows application.

4.16. Suitability for cloud deployment, automated CI/CD workflows

The software engineering world has gradually adopted open-source software due to the ease and possibility of integrating those tools in cloud systems and the ability of those tools to be incorporated on continuous integration and continuous delivery workflows which are typically run on cloud systems.

Given that both Metanorma and ShapeChange are open-source applications, both tools can easily be incorporated into continuous cloud workflows as demonstrated in the prototypes.

- In the case of PIM-to-MDS prototypes, the only manual step is the generation of the XMI export from EA for Metanorma consumption. Metanorma uses LutaML which directly parses the EA-proprietary profile of the XMI standard of the Object Management Group (OMG).
- In the case of PIM-to-PSM prototypes, the only manual step is the copying of the EA JAR file from EA to ShapeChange, in order to allow ShapeChange to access the EA-proprietary project (EAP) format.

The limitation of these manual steps presented by EA prevents the PIM being cloud-managed and used in dynamic generation of MDSes and PSMs.

Given that the choice of model authoring tool may be constrained, the non-standard formats used by those tools will force additional labor in establishing workflows, and need to be considered as part of tool evaluation when pursuing MDA.

4.17. Future work

4.17.1. General

This section describes possible work to be done in the future. These are in addition to the findings and recommendations discussed above, which should all be addressed.

4.17.2. SMART standards and models

Because the subject matter of MDS is modelled in information models, MBA makes it possible for documentation to be oriented to both human audiences and machine audiences, as human-readable or machine-readable artifacts.

As described in this document, standards documents themselves are represented as and governed by information models. This means that the supplementary truth in those standards — the guidance and normative content itself — can be made machine-readable and machine-addressable.

The BR SMART initiative (see Annex C) currently underway harnesses this approach to make standard documents themselves fully machine-readable, including granular semantic modelling of normative statements and requirements.

This granular modelling also enables for the exportation to a semantic web-based information representation like W3C RDF.

The benefits of this initiative can be harnessed by OGC to derive greater benefit from its standards, for example in automated testing.

4.17.3. Cross-reference integration between PIM and MDS

References to bibliography and terminological entry, as supplementary truth, are managed in the Metanorma environment, as is appropriate for an authoring environment.

This poses a challenge when cross-references to those entries, or to document content (supplementary truth), need to be made as part of the derived truth, i.e. the content authored within the model management tool, in such places as annotations.

In the case of UML Class diagrams, that arises in case the classes incorporate commentary and annotations with cross-references, as human-readable content.

Because the references the annotations cross-reference are defined externally, in Metanorma, there is a risk that the cross-references and the references will fall out of sync during authoring: if a reference is updated in Metanorma, the author may not find it is disrupting a reference from the PIM until much later. This makes managing cross-references more laborious than if both the supplementary and the derived truth were managed in the same authoring environment.

NOTE: Model-internal cross-references to other components of the same PIM, e.g. from one UML class to other UML classes, are much more likely to be well-defined through the tools used to manage the PIM itself, so they do not present the same level of challenge.

To manage external cross-references out of the PIM requires one of the following approaches:

- Use a PIM tool that supports external definitions of cross-references.

  Example 1: Sparx Systems Enterprise Architect UML class annotations can include Metanorma-formatted references to anchors that are defined in the Metanorma target document (e.g. `as defined in <<ISO-639>>`).

  - While this works, it tethers the PIM to a particular combination of authoring environment and definitions of anchors. This makes it more difficult to reuse PIM annotations in different documents, which might apply different anchor identifiers.

  - Reuse becomes even more difficult if the reference is to a clause in the target document: a naive document-internal reference may need to be changed to a bibliographic citation.

  - It also means that bibliography and terminology references need to be maintained separately in Metanorma and the UML authoring tool.

- Cross-references are not made machine-readable in the UML authoring tool at all, and are left as pure text.

  Example 2: In the PIM authoring tool, spell out references to standards documents as document titles. This approach was used during this prototype task.

  - While this avoids the problem, it also means that the document is semantically impoverished, and there is inconsistent presentation of cross-referenceable information between the derived truth and the supplementary truth in a standards document.

The proper solution to this challenge is tighter integration between Metanorma and PIM authoring tools.

Example 3: The PIM authoring tool and Metanorma could potentially share a bibliography plugin, or that Metanorma can import bibliographic information from the PIM (via LutaML) using a map that translates cross-references in PIM annotations into Metanorma cross-references.

4.17.4. ModSpec as PIM and ModSpec validation

Currently, ModSpec instances constitute supplementary truth in an MDS, as separately authored Metanorma content outside of the PIM. ModSpec instances can be encoded in Metanorma AsciiDoc or in the Metanorma-specific ModSpec-YAML format.

The Metanorma ModSpec authoring process today already provides some validation of ModSpec instances, such as:

- providing a fixed structure for entering of ModSpec model attributes
- warning if mandatory ModSpec model attributes are left empty
- confirming that conformance tests have matching requirements, and vice versa

Even so, during the development of the prototypes, challenges were encountered as detailed in Clause 4.11 due to inconsistencies found, including:

- inconsistent treatment of Conformance Test parts and Conformance Test steps (normative description differs from UML specification).
- additional unsanctioned ModSpec attributes presented in some OGC Standards that requires adaptation of MDS tool.
- usage of parameterization and conditionals in existing OGC Standards can differ from the ModSpec specification.
- lack of testable formal requirements for ModSpec compliance.

It is recommended to:

- Revise the ModSpec specification to resolve inconsistencies.
- Create a dedicated ModSpec authoring and validation tool for ModSpec instances to be used in a modular fashion and shared between standards, as originally intended. Validation of ModSpec instances will improve the quality of requirements and their adoption.
- Consider the possibility of machine-encoding of ModSpec for the purpose of automated testing and ModSpec validation, such as to ensure that conformance tests are satisfied by the model through automated testing.

In fact, Metanorma already provides a good basis of such implementation, by:

- fully implementing the ModSpec information model.
- accepting Metanorma AsciiDoc and YAML as input.
- providing some ModSpec validation.

- allowing the conversion of ModSpec into XML.

It would be possible for the Metanorma team to provide an open-source implementation for this purpose.

Once such a tool is developed, the ModSpec instances would be managed as an additional source of PIM primary truth (in addition to the original PIM), and integrated into the MDS transformation process just as is done with the PIM-to-PSM processes.

The resulting workflow would be automatic, and would not require manual input or adjustments.

5

MODEL SPECIFICATION TECHNOLOGIES

5 MODEL SPECIFICATION TECHNOLOGIES

5.1. General

Models are specified according to potentially one or more different technologies.

5.2. Unified Modeling Language (UML)

5.2.1. General

ISO/IEC 19501 specifies the UML modelling language, a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system.

UML specifies a set of methodologies for developing technical artifacts used in the design of a software system, ranging from business processes and system functions to programming language statements, database schemas, and reusable software components. UML is often used to develop domain-specific models (e.g., geospatial information) used in system development.

In this Testbed task, the use of UML in documenting a PIM relies on the following functionality defined in UML:

- For model definition, the definition of information models and their relationships, that contain human- and machine-readable components, and
- For class diagrams, the visual arrangement of UML class relationships intended for human consumption only.

5.2.2. Modelling capability

5.2.2.1. Basic modelling elements

UML provides three (3) basic modeling elements:

- **Package**
- **Class**, and

- **Property**

The usage of UML can extend beyond these modelling elements for these purposes:

- System developers may sometimes require additional modeling capabilities beyond those defined in the UML standard.
- System developers may need to attach a variety of types of information to models

  Example : To guide the derivation of technology-specific implementations.

5.2.2.2. Extensions

UML provides three (3) built-in extension mechanisms that enable new kinds of modeling elements to be added to the modeler's repertoire, as well as to attach "free-form" information to modeling elements. The extension mechanisms can be used separately or in combination to define new modeling elements that can have distinct semantics, characteristics, and notation relative to the built-in modeling elements specified by the UML metamodel.

These extension mechanisms are:

- **Stereotype**,
- **Tagged Value**, and
- **Constraint**.

5.2.2.3. Stereotype

A stereotype is an extension mechanism that is used to extend a UML metaclass by enabling the use of platform- or domain-specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass.

More than one stereotype may be simultaneously used to extend a UML metaclass.

Example 1: The «`FeatureType`» stereotype applied to a UML Classifier (class) might indicate that in an implementation a spatial representation will be assigned and spatial indexing should be implemented.

Example 2: The «`identifier`» stereotype applied to a UML Attribute property might indicate that the property value may be used as the Primary Key in a relational implementation.

5.2.2.4. Tagged value

A tagged value is an extension mechanism that is used to define (as a “tag definition”) a stereotype-specific characteristic that extends a UML model element (package, class, property) with additional information (as a “tagged value”).

Different stereotypes for a given UML metaclass may specify different sets of tag definitions. A given tag definition may be associated with more than one stereotype.

Example 1: A `definition` tagged value may be assigned to a model element in order to store a “concise definition of the element”

Example 2: A `version` tagged value may be assigned to store configuration management information for that element.

5.2.2.5. Constraints and Object Constraint Language (OCL)

A constraint is an extension mechanism that is used to express characteristics of a model that cannot be expressed in the UML class diagram notation. For example, a validation constraint that the allowed values of two properties interact and only certain patterns of values are allowed.

Constraints are usually expressed using the formal notation of ISO/IEC 19507:2012.

5.2.2.6. UML Profile

A collection of coordinated UML extensions may be documented in the form of a UML Profile. A UML Profile consists of a well-specified set of stereotypes, tagged values, and constraints that collectively specialize and tailor the UML for a specific domain (e.g., geospatial information modeling) or process.

A profile does not extend UML by revising the UML metamodel. Instead, it provides conventions for applying and specializing standard UML metamodel concepts to a particular environment or domain.

5.2.3. Application to geospatial standards (ISO 19100-series UML Profile)

5.2.3.1. ISO 19103:2015 Geographic information — Conceptual schema language

ISO 19103:2015 provides rules and guidelines for the use of a conceptual schema language to model geographic information, and specifies a profile of UML.

That profile includes six stereotypes:

- «Interface» (formerly «Type») is an abstract classifier with operations, attributes and associations, which can only inherit from or be inherited by other interfaces (or types).
- «DataType» is a set of properties that lack identity (independent existence and the possibility of side effects). A data type is a classifier with no operations, whose primary purpose is to hold information.
- «Union» is a type consisting of one and only one of several alternative datatypes (listed as member attributes); this is similar to a discriminated union in many programming languages.
- «Enumeration» is a fixed list of valid identifiers of named literal values. Attributes whose range type is an enumeration may only take values from the fixed list.
- «CodeList» is a flexible enumeration that uses string values for expressing a list of potential values. The allowed values are often held and managed using an online register.
- «Leaf» is a package that contains only classes (packages are disallowed).

The names of stereotypes are not case-sensitive. Stereotypes are essential in the derivation of PSMs from PIMs. Generally speaking, stereotypes can act as flags to determine how to create implementation models from abstract models.

The ISO 19103:2015 profile of UML also includes one tagged value:

- codeList, applies to stereotype «CodeList»: Code lists managed by a single external authority may carry a tagged value "codeList" whose value references the actual external code list. If the tagged value is set, only values from the referenced code list are valid.

The ISO 19103:2015 profile of UML is summarized in Figure 13.

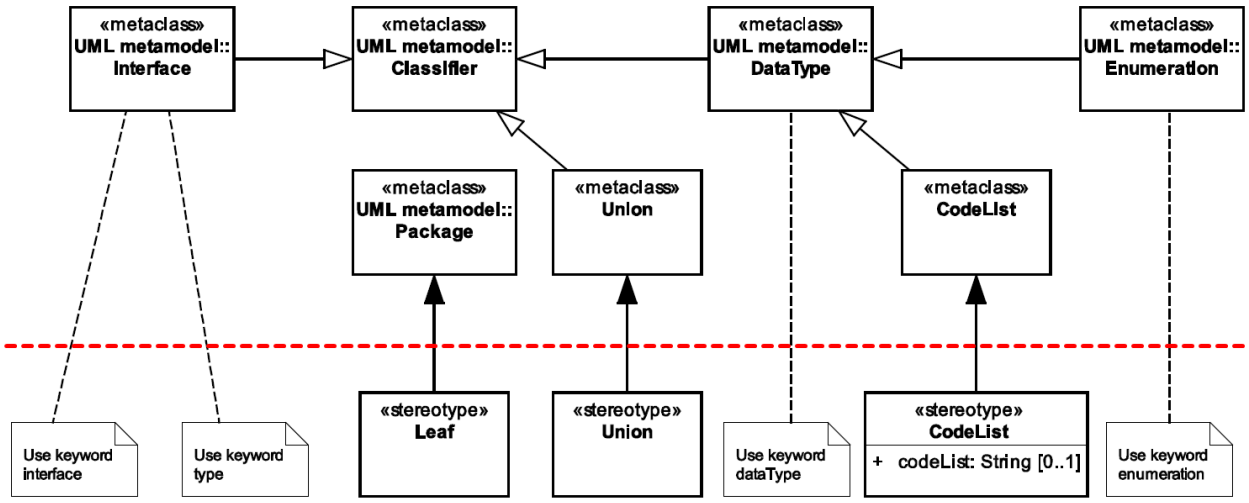


Figure 13 — ISO 19103:2015 stereotypes and keywords

5.2.3.2. ISO 19109:2015 Geographic information — Rules for application schema

ISO 19109:2015 defines rules for creating and documenting application schemas (conceptual schemas for data required by one or more applications), including principles for the definition of features, a fundamental unit of geographic information. As part of the general rules for application schemas it specifies the “General Feature Model” (GFM), the meta-model for application schemas.

The ISO 19109:2015 profile of UML that is used as the conceptual schema language for application schemas adds the critical «`ApplicationSchema`» (package) and «`FeatureType`» (class) stereotypes as well as three important tagged values that apply to both of these stereotypes:

designation	Natural language designator for the element to complement the name. Optional, with multiple designations allowed in order to support different languages.
definition	Concise definition of the element. One definition is mandatory. Additional definitions can be provided in multiple languages if required.
description	Description of the element, including information beyond that required for concise definition but which may assist in understanding its scope and application. Optional, with multiple descriptions allowed in order to support different languages.

The ISO 19109:2015 profile of UML is summarized in Figure 14:

Stereotype or keyword	UML metaclass	Constraints	Tagged values	Source
ApplicationSchema	Package		version	This International Standard, 8.2.3
			description catalogue-entry	This International Standard, 8.2.4
		Not nested in another applicationSchema package		This International Standard, 8.2.5
			language designation definition	This International Standard, 8.12
CodeList	Class			ISO 19103:—[5], 6.5.3, 6.10
dataType		Use in attributes or strong aggregation (composition)		ISO/IEC 19505-2:2012, 7.3.11 ISO 19103:—[6], 6.10
enumeration				ISO/IEC 19505-2:2012, 7.3.16 ISO 19103:—[7], 6.5.2, 6.10
Estimated	Property			This International Standard, 8.2.9
FeatureType	Class		description	This International Standard, 8.2.4, 8.2.6
			designation definition	This International Standard, 8.12
Leaf	Package	Cannot contain another package		ISO 19103:—[8], 6.10
Union	DataType			ISO 19103:—[9], 6.10
use				ISO/IEC 19505-2:2012, 7.3.40

Figure 14 — Summary of ISO 19109:2015 profile of UML

5.2.4. ISO 19118:2011 Geographic information — Encoding

ISO 19118:2011 specifies the requirements for defining encoding rules for use for the interchange of data that conform to the geographic information in the set of International Standards known as the "ISO 19100 series". It specifies requirements for creating encoding rules based on UML schemas, requirements for creating encoding services, and requirements for XML-based encoding rules for neutral interchange of data. It specifies a profile of UML that includes eight stereotypes, two of which are not previously defined similarly by either ISO 19103:2015 or ISO 19109:2015.

These are:

- «`BasicType`» is defined in ISO 19118:2011, Clause C.2.1.2:

  > "Defines a basic data type that has defined a canonical encoding." Additionally stated is that: "This canonical encoding may define how to represent values of

the type as bits in a memory location or as characters in a textual encoding. Examples of simple types are integer, float and string."

Also from ISO 19118:2011, Clause C.5.2.1.1:

"A class stereotyped «`BasicType`» shall be converted to a simpleType declaration in XML Schema. Any of the data types defined in XML Schema can be used as building blocks to define user-defined basic types. The encoding of the basic types shall follow the canonical representation defined in XML Schema Part 2: Datatypes." (W3C TR xmlschema-2)

+ NOTE: The different types are not clearly defined in ISO/TS 19103:2005 and neither is the «`BasicType`» stereotype used. The following declarations, therefore, follow a subset of the data type definitions in W3C TR xmlschema-2. Declared are the types: Number, Integer, Decimal, Real, Vector, Character, CharacterString, Date, Time, DateTime, Boolean, Logical, Probability, Binary, and UnlimitedInteger (where the symbol "*" is used to represent the infinite value).

- «`Interface`» From ISO 19118:2011, Clause C.2.1.2:

  "Defines a service interface and shall not be encoded."

  This definition is inconsistent with that of the subsequently published ISO 19103:2015. While this inconsistency may be useful in contexts where it is clear which definition applies, in general it is undesirable to overload the meanings of stereotypes within the OGC community, and in particular thereby coming into conflict with a stereotype specified in ISO 19103:2015.

  While the stereotype «`Interface`» as defined in ISO 19118:2011 can be (and is here) subsequently ignored, the stereotype «`BasicType`» is used in the CityGML 3.0 Conceptual Model where it results in difficulties given its tie to a specific encoding technology — XML Schema — and thus lack of true platform independence. The CityGML 3.0 Conceptual Model redefines the stereotype «`BasicType`» to mean "defines a basic data type", which is both circular and differs from that of ISO 19118:2011.

ShapeChange adopts a somewhat different definition for stereotype «`BasicType`» than either ISO 19118:2011 or the CityGML 3.0 Conceptual Model: (https://shapechange.net/app-schemas/uml-profile/#Stereotypes_of_classes)

Marks a so-called basic type, which can have a simple representation in one or more target encodings. It usually is a restriction of CharacterString or Number - or a structured version thereof (e.g., "one two three four" — or any other meaningful combination of such types that is supported by a given encoding).

5.2.4.1. ISO 19136-1:2020 Geographic information — Geography Markup Language (GML) — Part 1: Fundamentals

ISO 19136-1:2020 specifies an XML-based encoding (consistent with ISO 19118:2011) for the transport and storage of geographic information modelled in accordance with the conceptual modelling framework used in the ISO 19100 series of International Standards, including both

the spatial and non-spatial properties of geographic features. It specifies normative rules for the mapping of ISO 19109-conformant UML application schemas to corresponding GML application schemas based on XML Schema.

The ISO 19136-1:2020 profile of UML follows that of ISO 19109:2015 (and thus ISO 19103:2015) while adding numerous tagged values to packages, classes, and properties (Figure 15).

UML model element	Associated tagged values
Package	— documentation — xsdDocument — targetNamespace (only <<Application Schema>>) — xmlns (only <<Application Schema>>) — version (only <<Application Schema>>) — gmlProfileSchema (only <<Application Schema>>)
Class	— documentation — noPropertyType — byValuePropertyType — isCollection — asDictionary (only <<CodeList>>) — xmlSchemaType (only <<Type>>)
Attribute and association end	— documentation — sequenceNumber — inlineOrByReference — isMetadata

Figure 15 — Tagged values in the ISO 19136-1:2020 profile of UML

A valid UML Application Schema conforms to the conceptual schema rules specified in ISO 19103:2015 and the application schema rules specified in ISO 19109:2015.

A GML-ready UML Application Schema also conforms to the general encoding requirements of ISO 19136-1:2020. Those encoding requirements are summarized in Annex A. They identify required patterns and uses of UML elements, stereotypes, and tagged values based on the ISO 19109:2015 General Feature Model and the necessity of ensuring consistent application of the ISO 19136-1:2020 encoding rules.

5.2.5. Tools

In order to develop UML models for PIM-to-PSM and PIM-to-MDS purposes, it is necessary to define a toolchain that can interoperate from PIM to PSM and MDS stages for model transformation and downstream representations.

Various software tools exist to author UML models, and also to read such models.

5.2.5.1. Sparx Systems Enterprise Architect (EA)

Sparx Systems Enterprise Architect is widely used in OGC and ISO/TC 211 for the authoring and management of UML models.

Sparx Systems Enterprise Architect is historically a 32-bit Windows application:

- With the release of EA Version 15 a 64-bit Windows application may access an EA project file.
- With the upcoming release of Version 16 (in Beta mode as of January 2022), EA will be a 64-bit Windows application.

The UML artifacts it generates are the input to both ShapeChange, for generating PSMs (Clause 7.1), and to Metanorma and LutaML (Clause 7.2), for generating MDS.

5.2.5.2. ShapeChange

ShapeChange is an open-source tool for model conversions. It can read Sparx Systems Enterprise Architect project files using code provided by the Sparx Systems Enterprise Architect JAR file.

It is used to generate PSM outputs such as XML Schema, JSON Schema and RDF outputs from PIMs in the Sparx Systems Enterprise Architect format.

Through integration with EA, ShapeChange can access native EA artifacts directly, and then export the native EA project file into the XMI format (OMG XMI 2.5), then convert the exported XMI into the ShapeChange XML (SCXML) format.

5.2.5.3. LutaML and Metanorma

LutaML is a universal data model accessor that supports both UML and XMI, and is part of the Metanorma software suite which facilitates the generation of MDS documents.

Metanorma provides a LutaML plugin that allows Metanorma authors to access information models from various modelling languages, including UML and EXPRESS. Specifically, it supports parsing and accessing UML written in the XMI format, including the Sparx Systems Enterprise Architect profile of the XMI format.

LutaML can access UML packages stored in Sparx Systems Enterprise Architect project files by first exporting the UML package into the XMI format, of which the LutaML-XMI plugin can consume.

5.3. Resource Description Framework (RDF)

W3C RDF is a standard for representing graph data in terms of the Semantic Web, with Linked Data used to define subjects of description, as well as vocabularies for predicates, expressing attributes of entities and relations between entities, and predicate values, where these can be formally defined as discrete enumerations. It is widely used in semantically-informed description of data. RDF in MDA is extended through two common profiles in the Semantic Web domain:

- Web Ontology Language (W3C OWL, used to define ontologies and data domains and ranges; and
- Simple Knowledge Organization System (SKOS: W3C TR skos-reference), used to define hierarchical controlled vocabularies.

RDF can be used, like UML, to express PIMs which rely on notions of entities, inheritance, and attributes. The enhancement of RDF through OWL makes it more straightforward for RDF to model inheritance, while SKOS is used to model the constrained enumerations essential to MDA. Clause 9 reports on the testbed sub-task D143, where the use of RDF and OWL was investigated as an alternative to UML for driving MDA within OGC.

5.4. OGC Standard for Modular specifications

5.4.1. General

OGC 08-131r3 “The Specification Model — A Standard for Modular specifications”, also known as the ModSpec, contains requirements and conformance classes for writing standards to be used for any document whose eventual purpose is the specification of requirements for software, services or data structures.

OGC 08-131r3, Introduction helpfully uses an example from Thomas the Tank Engine, who strives to be a “really useful engine”, to explain its motivation in making standards useful.

> A standard’s usefulness and hence worth can be measured by:
>
> - The ease with which it can be implemented.
> - The number of times it has been implemented.
> - The number of times those implementations are used.
> - The ease with which those implementations interact.

- The number of times it has been extended through inclusion in other useful standards.

The intention of ModSpec is to enable interoperability of implementations, and the standard concludes that in order to validate requirements across implementations, it is necessary for conformance tests to be re-useable and organized in an effective manner. It achieves that by providing a standardized set of information models for these requirements and tests.

5.4.2. ModSpec models

ModSpec provides a set of information models towards this goal, as defined in OGC 08-131r3, Clause C.2, in UML, described below.

NOTE 1: The order of these described models differ from the ModSpec standard for clarity of conceptual coherence.

- Container:
  - Specification (OGC 08-131r3, Clause C.3). A specification is a container, a namespace for a set of ModSpec model instances. It represents a “standard”, such as an OGC or ISO standard.

- Related to specification:
  - Normative statement (OGC 08-131r3, Clause C.8). A normative statement.
  - Requirement (OGC 08-131r3, Clause C.9). A requirement is a normative statement that is required in an implementation. It links to at least one conformance test which, when passed, confirms the satisfaction of the requirement.
  - Recommendation (OGC 08-131r3, Clause C.10). A recommendation is a normative statement that presents optional but recommended functionality. There may or may not be a conformance test associated with it.
  - Permission (OGC 08-131r3, Clause 4.20). A permission is a normative statement that is entirely optional. While this model type is not directly defined in the ModSpec standard, it is described at OGC 08-131r3, Clause 4.20. There may or may not be a conformance test associated.
  - Requirements class (OGC 08-131r3, Clause C.6). A requirements class represents a coherent feature set. A requirements class can depend on other requirements classes and requirements modules, and applies to a standardization target type.
  - Requirements module (OGC 08-131r3, Clause C.7). A requirements module represents a coherent set of requirements needed by a feature. It is composed of a set of requirements.

- Related to verification:
  - Conformance suite (OGC 08-131r3, Clause C.4). A conformance suite is a set of conformance classes. It is intended to represent a set of conformance classes that represent a “feature set” intended to test an implementation.
  - Conformance class (OGC 08-131r3, Clause C.5). A conformance class is a set of conformance tests, and links to at least one requirements class. Conceptually it represents a “feature” of an implementation. Implementations may claim conformance to a “conformance class” for a feature that was validated.
  - Conformance test (OGC 08-131r3, Clause C.11). A conformance test represents a functional test intended to validate an implementation function. A conformance test is allowed to have separate parts and hierarchical steps (OGC 08-131r3, Clause C.11). A conformance test can be abstract, i.e. requires additional information to be executable.

    NOTE 2: See Annex D.1 for a discussion of the validity of test parts and test steps.
  - Abstract test. An abstract test is simply a conformance test that is abstract. While the term “abstract test” is not formally defined by ModSpec, it is heavily used in OGC Standards (and to that extent, standards published by ISO/TC 211).
  - Abstract test class. An abstract test class is a conformance class that only contains abstract tests. It is not formally defined by ModSpec but is used widely in OGC Standards.
  - Abstract test suite. An abstract test suite is a conformance suite that only contains abstract tests. It is not formally defined by ModSpec but is used widely in OGC Standards.

- Role:
  - Standardization target (OGC 08-131r3, Clause C.12). An entity to which some requirements of a standard apply.
  - Standardization target type (OGC 08-131r3, Clause C.13). The type of entity or set of entities to which the requirements of a standard apply.

5.4.3. Usage of ModSpec in OGC

According to the OGC Policy Directives, OGC Standards that contain requirements must have those requirements conform to OGC 08-131r3. Therefore, all OGC Standards are required to utilize ModSpec defined models in the representation of specification and verification mechanisms.

By using these information models, requirements from different OGC Standards can in effect be considered “modular”, i.e. these requirements can coexist in the same standard without being in conflict.

The ModSpec models themselves are abstract information models, and in order to use them, OGC Standards are closely gated by the OGC Standards team to have “instantiated” ModSpec models with the following requirements:

1. Each ModSpec model instance has a unique identifier in form of an OGC-wide unique URI
2. The linkages between the defined ModSpec model instances are validated to be correct and consistent.
3. The ModSpec model instances are published in the OGC Definitions Server.

Prior to this Testbed deliverable, the above steps and validation are often performed manually, which presents a heavy burden on both the standardization project team and the OGC editorial team.

In the course of this Testbed, we have found certain issues in the ModSpec standard that were resolved. Details are documented in Annex D.

5.5. Metanorma

5.5.1. General

As described in Clause 3.10.5, Metanorma is an open-source framework for creating and publishing standardization artifacts with the focus on semantic authoring and flexible output support.

Metanorma provides a model-based documentation system and prioritizes automation.

Overall, Metanorma provides the following items:

- a set of standard document metamodels (ISO/AWI 36100), called “Metanorma StanDoc”, that allows different standardization bodies to create their own standardized document model for their standard documents;
- a standard XML schema (ISO 36300), as machine-readable standardization documents, that is the serialized form of the document models mentioned above; and
- a publishing toolchain that enables authors of standard documents to handle their documents from authoring to publishing in an end-to-end, “author-to-publish” fashion.

The canonical output of Metanorma is an XML file that represents an ISO/AWI 36100 model tree.

In addition, the Metanorma technology suite also provides:

- standards metadata specification capability, based on ISO 36200, a set of metamodels for the development of organization-specific bibliographic item models from ISO 690, the international standard for citations and references
- ability to encode concepts (and terms and definitions) in interoperable formats in accordance with the ISO 10241-1 and ISO 704 standards
- ability to directly access certain information models types through the Ribose LutaML model parsing engine, including:
    - EXPRESS (ISO 10303-11)
    - OMG UML within OMG XMI (OMG UML 2.5, OMG XMI 2.5) and Enterprise Architect generated XMI
    - XSD (W3C TR xmlschema-1)
- ability to author UML models through open-source model specification tools including Ribose LutaML or PlantUML.

With the above functionality, Metanorma is often used as an annotation system on information models.

Example : ISO/TC 184/SC 4/TF 1 uses Metanorma to annotate EXPRESS schemas resulting in model-derived documentation of EXPRESS schemas.

5.5.2. Usage in OGC

“Metanorma for OGC” is the implementation of Metanorma for OGC. It has been approved as an official way to publish new OGC Standard documents since 2021-09-17, with Metanorma-based document templates subsequently approved by the OGC Document SubCommittee on 2022-02-25.

Metanorma for OGC documents are created in the Metanorma AsciiDoc format. Metanorma AsciiDoc is a textual syntax for preparing a ISO/AWI 36100 compliant document model tree which can be rendered in a variety of presentation formats. In OGC, the supported rendering formats are PDF, HTML, Microsoft Word, and ISO/AWI 36100 XML.

Metanorma for OGC differs from typical Metanorma in the following critical ways:

- supports specification of OGC Standards metadata, including document types, stages, identifiers and authorship;
- supports specification of OGC ModSpec (OGC 08-131r3) model instances through a specialized syntax.

Figure 16 shows the range of models used in Metanorma, including the OGC-specific use of OGC ModSpec.

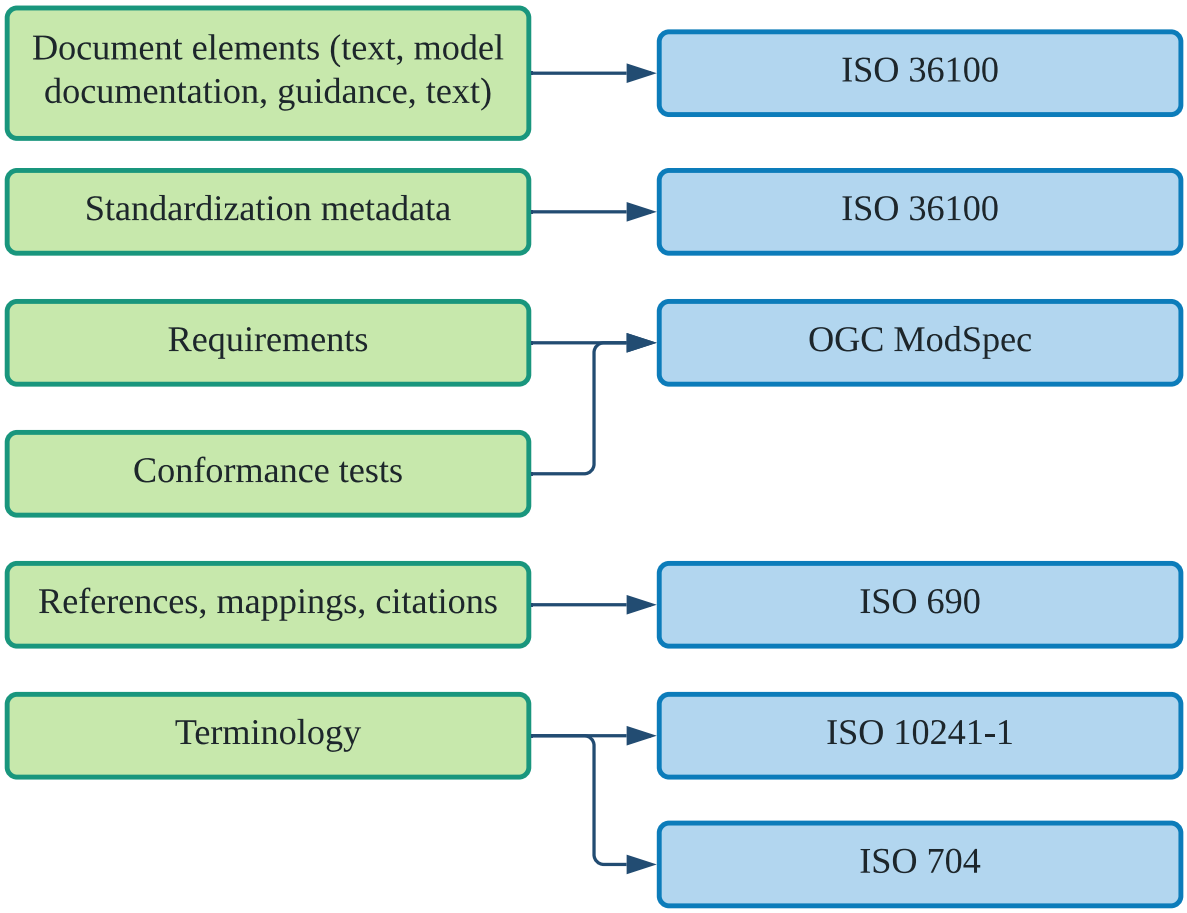


Figure 16 — Models used in Metanorma

5.5.3. Specification of ModSpec model instances

5.5.3.1. General

OGC ModSpec (OGC 08-131r3) specifies a set of UML models that allow a unified mechanism to describe requirements in OGC Standards.

Metanorma for OGC provides a special syntax for the encoding and embedding of requirements compliant to OGC ModSpec, for the exporting of machine-readable requirements as well as ModSpec compliant rendering. These models can then be treated and presented in a uniform manner.

All models in ModSpec are supported, as described in Clause 5.4.2.

5.5.3.2. Basic syntax

The basic syntax for specifying a ModSpec instance within Metanorma is shown in the following Example.

Example — ModSpec basic syntax:

```
[requirement]
====
[%metadata]
label:: /req/core

Requirement text.
====
```

Where:

- `requirement` specifies the type of ModSpec class
- `label` specifies the OGC identifier of the ModSpec instance
- `Requirement text.` represents the description of the requirement.

NOTE: Full syntax used for specifying ModSpec instances is documented in Annex E.

5.5.3.3. ModSpec types

The ModSpec model types supported by Metanorma can be specified using the syntax shown in Figure 17.

```
[recommendation]
[permission]
[requirements_class]
[conformance_test]
[conformance_class]
[abstract_test]
// [abstract_test] is short for [conformance_test,%abstract]
```

Figure 17 — ModSpec types supported

NOTE: The full syntax used for specifying ModSpec instances is documented in Annex E.

5.5.3.4. Examples of rendered ModSpec instances

The following examples demonstrate how ModSpec instances are represented in OGC Standards.

Example 1 — ModSpec example: OGC 20-010 (CityGML 3.0) Requirements Class 1:

```
[requirements_class]
====
[%metadata]
```

```
label:: http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core
subject:: Implementation Specification

inherit:[<<iso19103>>]
inherit:[<<iso19107>>]
inherit:[<<iso19109>>]
inherit:[<<iso19111>>]
inherit:[<<iso19123>>]
inherit:[<<xal2,OASIS xAL v3.0>>]
inherit:[req/req-class-core]
====
```

REQUIREMENTS CLASS 1	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core	
Target type	Implementation Specification
Dependency	ISO 19103:2015
Dependency	ISO 19107:2003
Dependency	ISO 19109:2015
Dependency	ISO 19111:2019
Dependency	ISO 19123:2005
Dependency	OASIS xAL v3.0

Example 2 — ModSpec example: OGC 20-010 (CityGML 3.0) Requirement 1:

```
[requirement,type="general",label="/req/core/classes"]
====
For each UML class defined or referenced in the Core Package:

[.component,class=part]
--
The Implementation Specification SHALL contain an element which represents the
same concept as that defined for the UML class.
--

[.component,class=part]
--
The Implementation Specification SHALL represent associations with the same
source, target, direction, roles, and multiplicities as those of the UML class.
--

[.component,class=part]
--
The implementation Specification SHALL represent the attributes of the UML
class including the name, definition, type, and multiplicity.
--

[.component,class=part]
--
```

```
The implementation Specification SHALL represent the attributes of all
superclasses of the UML class including the name, definition, type, and
multiplicity.
--

[.component,class=part]
--
The implementation Specification SHALL represent the associations of all
superclasses of the UML class including the source, target, direction, roles,
and multiplicity.
--

[.component,class=part]
--
The Implementation Specification SHALL specify how an implementation observes
all constraints the Conceptual Model imposes on the UML class.
--
====
```

REQUIREMENT 1	
/req/core/classes	
For each UML class defined or referenced in the Core Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and multiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how an implementation observes all constraints the Conceptual Model imposes on the UML class.

Example 3 — ModSpec example: OGC 20-010 (CityGML 3.0) Permission 1:

```
[permission,label="/per/core/classes"]
====
For each UML class defined or referenced in CityGML Conceptual Model:

[.component,class=part]
--
```

```
An Implementation Specification MAY represent that class as a null class with
no
attributes, associations, or definition.
--

[.component,class=part]
--
An Implementation Specification MAY represent an association of the UML class
with a null association.
--

[.component,class=part]
--
An Implementation Specification MAY represent an attribute of the UML class
with
a null attribute.
--

[.component,class=part]
--
Whenever a null element is used to represent a concept from the Conceptual
Model, the Implementation Specification SHOULD document that mapping and
provide
an explanation for why that concept was not implemented.
--
====
```

PERMISSION 1	
/per/core/classes	
For each UML class defined or referenced in CityGML Conceptual Model:	
A	An Implementation Specification MAY represent that class as a null class with no attributes, associations, or definition.
B	An Implementation Specification MAY represent an association of the UML class with a null association.
C	An Implementation Specification MAY represent an attribute of the UML class with a null attribute.
D	Whenever a null element is used to represent a concept from the Conceptual Model, the Implementation Specification SHOULD document that mapping and provide an explanation for why that concept was not implemented.

Example 4 — ModSpec example: OGC 20-010 (CityGML 3.0) Requirement 2:

```
[requirement,type="general",label="/req/core/isorestrictions"]
====
ISO classes used in the CityGML Conceptual Model are subject to the following
restrictions:

[.component,class=part]
--
Classes derived from the GM_Solid class (ISO 19107) SHALL only include exterior
boundaries. (The `interior` association on the GM_SolidBoundary shall not be
defined)
```

```
--
====
```

REQUIREMENT 2	
/req/core/isorestrictions	
ISO classes used in the CityGML Conceptual Model are subject to the following restrictions:	
A	Classes derived from the GM_Solid class (ISO 19107) SHALL only include exterior boundaries. (The `interior` association on the GM_SolidBoundary shall not be defined)

Example 5 — ModSpec example: OGC 20-010 (CityGML 3.0) Abstract Test 2, which corresponds to Requirement 2:

```
[abstract_test]
====
[%metadata]
label:: /ats/core/isorestrictions
subject:: <<req_Core_iso-restrictions,/req/core/isorestrictions>>
[.component,class=test-purpose]
--
To validate that none of the restrictions which the CityGML Conceptual Model
imposes on ISO classes are violated by an Implementation Specification.
--

[.component,class=test method type]
--
Manual Inspection
--

[.component,class=test method]
=====
[.component,class=step]
--
For each instance of the GM_Solid class, validate that there are no interior
boundaries associated with that instance.
--

[.component,class=step]
--
For each instance of a class descended from the GM_Solid class, validate that
there are no interior boundaries associated with that instance.
--
=====
====
```

ABSTRACT TEST A.2	
/ats/core/isorestrictions	
Requirement	/req/core/isorestrictions
Test purpose	To validate that none of the restrictions which the CityGML Conceptual Model imposes on ISO classes are violated by an Implementation Specification.
Test method type	Manual Inspection
Test method	1. For each instance of the GM_Solid class, validate that there are no interior boundaries associated with that instance. 2. For each instance of a class descended from the GM_Solid class, validate that there are no interior boundaries associated with that instance.

5.5.4. Specification of terminology entries

Metanorma supports the specification of terminology entries (concepts) as instances of the information models described in ISO 10241-1 and ISO 704.

NOTE 1: The information models used are available in LutaML format at https://github.com/glossarist/concept-model.

Concepts can be sourced from other standards the OGC definitions server as well as other termbases, such as the IEV.

Example 1 — Concept example: OGC 20-010 (CityGML 3.0) 4.1.6, sourced from ISO 19101-1:

```
==== conceptual model

model that defines concepts of a universe of discourse

[.source]
<<iso19101-1,clause=4.1.5>>
```

4.1.6. conceptual model

model that defines concepts of a universe of discourse

[**SOURCE:** ISO 19101-1:2014, Clause 4.1.5]

Example 2 — Concept example: OGC 20-010 (CityGML 3.0) 4.1.8, sourced from the OGC Definitions Server:

```
==== Implementation Specification

Specified on the OGC Document Types Register

[.source]
<<OGCDTR,locality:URI="http://www.opengis.net/def/doc-type/is">>
```

4.1.8. Implementation Specification

Specified on the OGC Document Types Register

[**SOURCE:** OGC Document Types Register, URI http://www.opengis.net/def/doc-type/is]

NOTE 2: See detailed syntax at https://www.metanorma.org/author/ogc/topics/markup/.

OGC Standards often have an informative “Glossary” that presents definitions of terms. An example of term definition within the “Glossary” is given here.

Example 3 — Concept example: OGC 20-010 (CityGML 3.0) C.1.2.3, sourced from ISO 19111:

```
=== CC_CoordinateOperation

A mathematical operation on coordinates that transforms or converts coordinates
to another coordinate reference system.

[.source]
<<iso19111>>
```

C.1.2.3. CC_CoordinateOperation

A mathematical operation on coordinates that transforms or converts coordinates to another coordinate reference system.

[**SOURCE:** ISO 19111:2019]

5.5.5. Specification of bibliographic references

Metanorma supports the specification of bibliographic entries as instances of the information models described in ISO 690.

NOTE 1: The information models used are available in LutaML format at https://github.com/relaton/relaton-models.

Example 1 — Bibliographic item example from OGC 20-010 (CityGML 3.0) of "OGC 15-001r4":

```
[bibliography]
== Bibliography

* [[[three-dps_citation,OGC 15-001r4]]], Hagedorn, B., Thum, S., Reitz, T., Coors, V., Gutbell, R.: _OGC® 3D Portrayal Service 1.0_, Open Geospatial Consortium, Available from https://docs.opengeospatial.org/is/15-001r4/15-001r4.html[OGC Doc. 15-001r4].
```

BIBLIOGRAPHY

1. Benjamin Hagedorn, Simon Thum, Thorsten Reitz, Voker Coors, Ralf Gutbell: OGC 15-001r4, *OGC® 3D Portrayal Service 1.0*. Open Geospatial Consortium (2017). http://docs.opengeospatial.org/is/15-001r4/15-001r4.html

Metanorma provides auto-fetch functionality for a number of publications from Standards Development Organizations (SDO), including OGC.

The above example will be still rendered identically given the following information, as the relevant metadata about the publication identified is retrieved automatically.

Example 2 — Bibliographic item example from OGC 20-010 (CityGML 3.0) of "OGC 15-001r4" with auto-fetch:

```
[bibliography]
== Bibliography

* [[[three-dps_citation,OGC 15-001r4]]]
```

BIBLIOGRAPHY

1. Benjamin Hagedorn, Simon Thum, Thorsten Reitz, Voker Coors, Ralf Gutbell: OGC 15-001r4, *OGC® 3D Portrayal Service 1.0*. Open Geospatial Consortium (2017). http://docs.opengeospatial.org/is/15-001r4/15-001r4.html

NOTE 2: For more details, see https://www.metanorma.org/author/topics/document-format/bibliography/ .

6

MODEL USE TECHNOLOGIES

6 MODEL USE TECHNOLOGIES

6.1. General

Three platform-specific technologies were investigated as targets for application of the MDA paradigm to the preparation of Model-Driven Standards (MDS).

XML Schema, JSON Schema, and Resource Description Framework are alternative formats for use in machine-to-machine data exchange. Only the CityGML 3.0 Conceptual Model was evaluated for these three target technologies. In all three cases a ShapeChange-based PIM-to-PSM transformation process was employed.

6.2. XML Schema

XML Schema (https://www.w3.org/TR/xmlschema-1/) describes a class of XML documents by using schema components to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content and attributes and their values.

Given a PIM expressed using the ISO 19103:2015 profile of UML, ISO 19118:2011 “Geographic information — Encoding” specifies the requirements for defining encoding rules for use for the interchange of data that conform to the geographic information in the set of International Standards known as the “ISO 19100 series”. It specifies requirements for creating encoding rules based on UML schemas, requirements for creating encoding services, and requirements for XML-based encoding rules for neutral interchange of data. In its Annex A, an XML-based encoding rule identifies two well-known use cases in the ISO 19100-series of standards incorporating conceptual schemas.

- ISO/TS 19139 specifies an XML-based encoding rule for conceptual schemas specifying types that describe geographic resources, e.g. metadata according to ISO 19115:2003 and feature catalogues according to ISO 19110.
- ISO 19136:2007 specifies an XML-based encoding rule for ISO 19109:2015-conformant application schemas that can be represented using a restricted profile of UML that allows for a conversion to XML Schema. The encoding rule has mainly been developed for the purpose of application schemas specifying feature types and their properties.
    - ISO 19136-1:2020 “Geographic information — Geography Markup Language (GML) – Part 1: Fundamentals” specifies that a valid UML Application Schema input to the mapping specified in its Annex E.2 Encoding Rules shall conform to the applicable requirements of ISO 19118:2011, Annex A, “XML-based encoding rule”.

A GML-ready UML Application Schema also conforms to the general encoding requirements of ISO 19136-1:2020. Those encoding requirements are summarized in Annex A. They identify required patterns and uses of UML elements, stereotypes, and tagged values based on the ISO 19109:2015 General Feature Model and the necessity of ensuring consistent application of the ISO 19136-1:2020 encoding rules.

ISO 19136-1:2020, Annex E.2.4 "Conversion rules" specifies how XML Schema documents (XSD files) shall be derived from an application schema expressed in UML in accordance with ISO 19109:2015. Figure 18 summarizes those rules.

Table: UML → GML application schema overview	
UML application schema	**GML application schema**
Package	One XML Schema document per package (default mapping)
<<Application Schema>>	XML Schema document
<<DataType>>	Global element, whose content model is a globally scoped XML Schema complexType, property type
<<Enumeration>>	Restriction of xsd:string with enumeration values
<<CodeList>>	Union of an enumeration and a pattern (default mapping, an alternative mapping is a reference to a dictionary)
<<Union>>	Choice group whose members are GML objects or features, or objects corresponding to DataTypes
<<FeatureType>>	Global element, whose content model is a globally scoped XML Schema type derived by direct/indirect extension of gml:AbstractFeatureType, property type
No stereotype or <<Type>>	Global element, whose content model is a globally scoped XML Schema type derived by direct/indirect extension of gml:AbstractGMLType, property type
Operations	Not encoded
Attribute	local xsd:element, the type is either a property type (if the type is a complex type) or a simple type.
Association role	local xsd:element, the type is always a property type (only named and navigable roles)
General OCL constraints	Not encoded

Figure 18 — ISO 19136-1:2020 GML application schema encoding overview

ISO 19136-1:2020 does not specify an encoding rule for UML Association Classes. Instead, ISO 19136-2:2015 "Geographic information — Geography Markup Language (GML) — Part 2: Extended schemas and encoding rules" specifies a conversion rule that results in an "intermediate" class, which may then be serialized using the standard UML-to-GML encoding rule. The original model and then the result of applying the conversion rule are illustrated in Figure 19 and Figure 20.

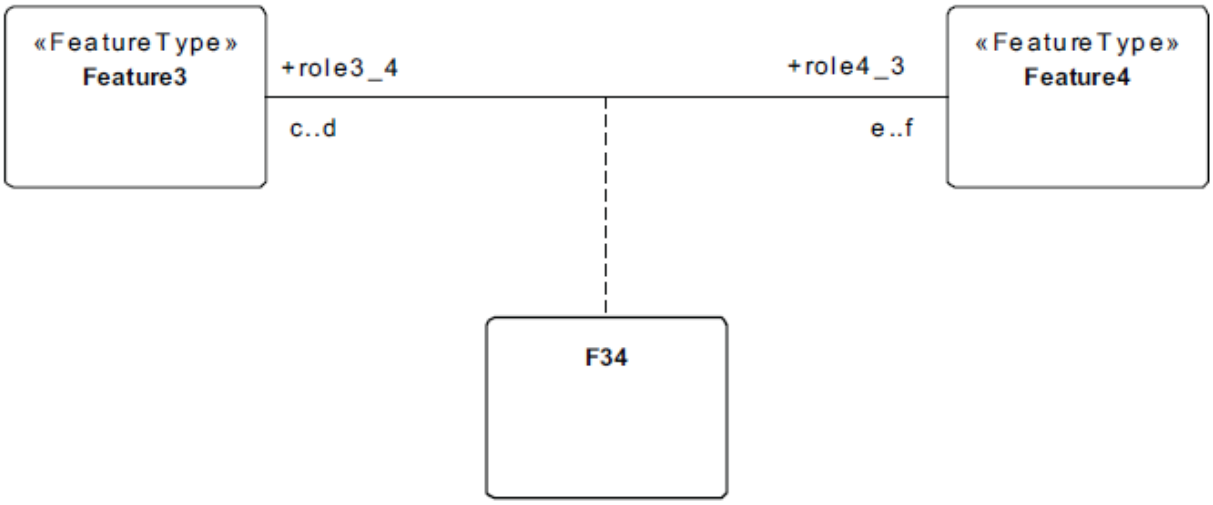


Figure 19 — Model with association class

Figure 20 — Model after conversion rule applied

6.3. JSON Schema

JavaScript Object Notation (JSON: RFC 8259) is a lightweight data-interchange format based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition — December 1999 (see: RFC 7159). Like eXtensible Markup Language (XML), it is text-based and easy for humans to read and write. It has a simple structure based on collections of name/value pairs and ordered lists (e.g., array, vector, list, or sequence).

GeoJSON (RFC 7946) is a geospatial data interchange format based on JSON. It defines several types of JSON objects and the manner in which they are combined to represent data about geographic features, their properties, and their spatial extents. GeoJSON uses a geographic coordinate reference system, World Geodetic System 1984, and units of decimal degrees.

JSON Schema, analogous to XML Schema, is a language for specifying the structure of JSON data. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data (see: Internet-Draft draft-bhutton-json-schema-00 and JSON Schema). JSON Schema uses keywords to assert constraints on JSON instances or annotate those instances with additional information. Additional keywords are used to apply assertions and annotations to more complex JSON data structures, or based on some sort of condition.

Unlike XML Schema which was first standardized in 2004, JSON Schema remains a work-in-progress by the Internet Engineering Task Force (IETF). The current draft is “2020-12”; this task

used draft “2019-09” as supported by the ShapeChange tool. Changes subsequent to draft “2019-09” are documented at: https://json-schema.org/draft/2020-12/release-notes.html No incompatible changes have been identified. Most JSON schema validators support either the earlier “draft-07” or draft “2019-09”. Changes subsequent to “draft-07” are documented at: https://json-schema.org/draft/2019-09/release-notes.html

A JSON Schema has been defined for GeoJSON (see: https://geojson.org/schema/Feature.json and https://github.com/geojson/schema). That schema was employed as part of this task. Unfortunately, there are currently no JSON Schemas defined for important ISO 19100-series conceptual schema standards.

In June 2021 the OGC chartered a Features and Geometries JSON SWG to develop a JSON encoding for geospatial feature data (see: https://www.ogc.org/projects/groups/featgeojsonswg). The OGC Features and Geometries JSON candidate standard extends the GeoJSON standard, adding minimal extensions to support additional concepts that are important for the wider geospatial community and the OGC API standards. The evolving draft is accessible at: https://github.com/opengeospatial/ogc-feat-geo-json. Approaches to extending GeoJSON being considered there have informed aspects of this task.

7

MODEL TRANSFORMATION APPROACHES

7 MODEL TRANSFORMATION APPROACHES

7.1. ShapeChange

7.1.1. General

ShapeChange is a powerful, extensible capability with many configuration choice-points involved in designing a set of Conceptual Model transformations and technology targets. It is used by OGC to generate PSMs from PIMs. While individual choice-points and their controls are generally well-documented, ShapeChange has a steep learning curve for the novice. In this ER we provide some "cookbook" recipes to assist new users for specific use-cases.

ShapeChange automatically validates the specified configuration on startup (see: https://shapechange.net/get-started/config/validation-of-the-configuration/). This prevents configuration mistakes that would otherwise only be noticed during or after processing a model.

ShapeChange includes a powerful logging capability that records both Warnings and Errors associated with each rule application. Errors require corrections to the input model, or possibly the processing configuration; Warnings may be either in the nature of advice, or an issue that ShapeChange may attempt to resolve by introducing a model assumption and then proceeding. In many cases that assumption is best avoided by improving the model so that it is unambiguous.

As a deterministic rule-based system ShapeChange requires significant internal consistency in an input Conceptual Model for best outcome; the quality of the resulting PSM is directly determined by the quality and completeness of the PIM – particularly the proper application of UML Stereotypes and Tagged Values. Such consistencies may not be present in existing models that were developed for the sole purpose of diagrammatic presentation in standards documents; such models may require enhancement in order to be useful for deriving corresponding PSMs.

ShapeChange includes the ability to verify the correctness of a Conceptual Model with respect to the requirements of UML (https://www.omg.org/spec/UML/2.5.1/About-UML/), ISO 19103:2015 (Conceptual schema language), and ISO/IEC 19507:2012 (OCL); it is thus useful as a PIM validation tool even in circumstances in which derivation of technology-specific implementation specifications is not intended.

7.1.2. Usage

ShapeChange is an open-source implementation of the MDA methodology that has emerged as a key technological underpinning in many geospatial data modeling environments where platform-specific data exchange requirements have been separated from platform-independent domain (content) modeling requirements.

MDA techniques have been used sporadically to develop OGC and ISO/TC 211 standards; ShapeChange has been instrumental in many of those efforts, although effort-specific documentation has been generally sparse. Engineering Reports documenting OGC ShapeChange initiatives going back to OGC Testbed-2 (OGC 04-100) can be found at https://shapechange.net/about/background-documents/

ShapeChange is a portable Java-based application. It has no native graphical interface. All control of ShapeChange processing is achieved through the use of configuration files that are XML Instance documents. The location of the key configuration file is the only ShapeChange-specific command-line parameter. Log files are also XML instance documents (and, for convenience, corresponding HTML files are generated for ready use in a browser where they enable presentation-filtering).

ShapeChange is documented at https://shapechange.net/. Its source code repository is established at https://github.com/ShapeChange/ShapeChange. It may be extended: https://shapechange.net/get-started/how-to-extend-shapechange/

A typical ShapeChange use-case begins with opening and reading a UML model from a Sparx Systems Enterprise Architect (EA) repository. In order to do so, EA must be installed on the computer that executes ShapeChange.

Each EA installation comes with an `SSJavaCOM.dll` and/or `SSJavaCOM64.dll`, which must be installed correctly in order for ShapeChange to access local EA repositories. For further installation details, see: http://shapechange.net/get-started/

In the case that ShapeChange is intended to be executed other than under Microsoft Windows with an EA repository, there are means to losslessly convert a Conceptual Model stored in an EA repository to the portable ShapeChange XML (SCXML) format for use in other environments (e.g., MacOS or Linux). There are also means to losslessly convert a Conceptual Model stored in an SCXML file into the EA repository format (under Microsoft Windows with an EA installation present).

7.1.3. UML profiles

ShapeChange supports the ISO 19100-series of UML Profiles as well as additional package, class, and property stereotypes; see: https://shapechange.net/app-schemas/uml-profile/#Stereotypes

ShapeChange supports an extensive range of tagged values, well beyond the base set specified in ISO 19109:2015 and ISO 19136-1:2020; see: https://shapechange.net/app-schemas/uml-profile/#Tagged_Values

OCL Constraints (a restricted subset of the full language) are parsed and used for the purpose of generating Schematron (ISO/IEC 19757-3 “Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation using Schematron”) assertions; see: https://shapechange.net/targets/xsd/extensions/ocl/

7.1.4. Descriptors

It is common practice to include a variety of descriptive information about elements in a UML model, e.g., aliases, codes, definitions, or descriptions. In order for ShapeChange transformations or targets to process each of these types of information distinctly they must be individually documented. UML tagged values are an ideal mechanism for documenting such information, however their application and use in UML modeling tools tends to be inconveniently supported and erratically employed by modelers.

UML modeling tools typically provide a single “notes” field whose value is prominently displayed and easily edited. Sparx Systems Enterprise Architect (EA) includes both a “notes” field and a separate “alias” field. A panel displaying the “notes” field can be pinned to the EA user interface for ready access whenever a model element is selected. In consequence of this field ubiquity, it is common practice in some modeling communities to populate the “notes” field using fixed separators in order to distinguish the different descriptors that have been combined into a single structured string for documenting descriptive information about an element.

NOTE: This means the EA “notes” field should not be used for human-readable annotations as well: a downstream processing system like LutaML will have difficulty establishing whether the “notes” field contains machine-readable descriptors or a human-readable annotation.

The ISO/TC 211 Harmonized Model Maintenance Group (HMMG: https://github.com/ISO-TC211/HMMG) uses the all-in-one approach based on the “notes” field rather than discrete tagged values. In some cases (e.g., package “ISO 19170 Discrete global grid systems”) text styling/formatting is introduced in order to improve the presentation within the EA element editor.

To support both tagged value and structured “notes” approaches, ShapeChange implements a set of descriptors for commonly types of element information (e.g., “alias”, “definition”, “description”, “documentation”, “example” and “primaryCode”); see: https://shapechange.net/get-started/config/input/#Descriptor_sources

ShapeChange provides mechanisms to parse the EA element “notes” into descriptors as well as to load descriptors from user-specified tagged values. A ShapeChange transformer (https://shapechange.net/transformations/descriptor-transformer/) may be used to manipulate descriptors during model processing. Target processors employ these descriptors in encoding-specific manners.

A simple ShapeChange configuration to load descriptors from an EA project file would include the following lines:

```
<descriptorSources>
  <DescriptorSource descriptor="documentation" source="ea:notes"/>
  <DescriptorSource descriptor="alias" source="ea:alias"/>
</descriptorSources>
```

Figure 21 — ShapeChange configuration to load descriptors from an EA project file

7.1.5. Rules

ShapeChange is a rule-based tool that manipulates an in-memory representation of a UML (class diagram) model. A rule specifies an action to be taken when matching a pattern in a Concept Model. Sets of rules are used either to transform one UML model to another (see: https://shapechange.net/transformations/#Rules), or, given a UML model, to generate a target encoding (in general see: https://shapechange.net/targets/#Encoding_Rules; for the XML Schema target in particular, see: https://shapechange.net/targets/xsd/#Specifying_Encoding_Rules and https://shapechange.net/targets/xsd/extensions/#Using_the_non-standard_conversion_rules).

For example, in support of the platform-specific GML 3.2 encoding ShapeChange specifies twenty encoding rules based on the encoding requirements of ISO 19136-1:2020. Those encoding requirements are summarized in Annex A: UML-to-GML application schema encoding rules. These encoding rules are collectively referenced within ShapeChange as "iso19136_2007". ShapeChange can be configured to include additional rules (e.g., `rule-xsd-cls-basictype-list` was added when processing the revised CityGML 3.0 Conceptual Model) as well as, in some cases, controlling the behavior of individual rules using configuration parameters.

Table 4 — Rules used to encode GML 3.2

RULE	DESCRIPTION
`rule-xsd-all-naming-gml`	Use the naming strategy for schema components as specified by GML 3.2
`rule-xsd-pkg-contained-packages`	For packages contained directly or indirectly in an application schema and which are converted to a separate XML Schema Document, add imports and includes
`rule-xsd-pkg-dependencies`	For dependencies from an application schema to another schema, add imports
`rule-xsd-pkg-gmlProfileSchema`	Include gmlProfileSchema appinfo, if the tagged value is set on the application schema package
`rule-xsd-cls-object-element`	Create global object elements for feature types, classes, data types and union types
`rule-xsd-cls-type`	Create global types for the content model of feature types, classes, data types and union types
`rule-xsd-cls-union-as-choice`	Properties of union types are converted to local property elements in a choice block
`rule-xsd-cls-unknown-as-object`	Treat classes of unknown characteristics as object classes
`rule-xsd-cls-property-type`	Create global property types for feature types, classes, data types and union types
`rule-xsd-cls-local-properties`	Create local property elements for properties in feature types, classes, data types and union types

RULE	DESCRIPTION
`rule-xsd-cls-sequence`	Properties of feature types, classes and data types are converted to local property elements in a sequence block
`rule-xsd-cls-global-enumeration`	Convert enumerations to global types
`rule-xsd-prop-inlineOrByReference`	Take tagged value inlineOrByReference on properties into account when setting the type of a property element
`rule-xsd-prop-reverseProperty`	Include reversePropertyName appinfo, if applicable
`rule-xsd-prop-targetElement`	Include targetElement appinfo, if a property is by Reference
`rule-xsd-cls-noPropertyType`	If the tagged value noPropertyType of a feature type, class, data type or union type is `true`, the property type creation is suppressed
`rule-xsd-cls-byValuePropertyType`	If the tagged value byValuePropertyType of a feature type or class is `true`, an additional by-value property type is created
`rule-xsd-cls-standard-gml-property-types`	Reuse property types or create anonymous property types according to GML 3.2
`rule-xsd-cls-codelist-asDictionary`	If the tagged value asDictionary of a code list is not `true`, create a global type for the code list; otherwise reference code list values using `gml:CodeType`
`rule-xsd-prop-defaultCodeSpace`	Include defaultCodeSpace appinfo, if applicable

The ShapeChange XML Schema target natively supports four sets of encoding rules (for GML 3.2, GML 3.3, ISO 19115 metadata and ISO 19110 feature catalogues, and Sensor Web Enablement (SWE) Common Data Model 2.0); see: https://shapechange.net/targets/xsd/#Encoding_Rules .

Additional well-known sets of encoding rules are documented in https://shapechange.net/resources/config/StandardRules.xml .

The set of encoding rules to apply in a model conversion may be provided in two ways: as a conversion-wide default (specified in the project configuration file), or as tagged values on individual model elements to control the conversion of those specific elements. To identify the applicable set of encoding rules for a model element (package, class, or property), a tagged value `xsdEncodingRule` may be specified on that element. The application of the specified set of encoding rules follows the usual containment rules of UML where, e.g., a specification applied to a package also applies to its contained classes and their properties, unless overridden by a different value assigned to one of those elements.

If the tagged value `xsdEncodingRule` is not specified anywhere in the UML model, then the ShapeChange `defaultEncodingRule` parameter specified for the target encoding in the project configuration file is applied. In many cases a single set of encoding rules will apply throughout a

model; however complex models incorporating components from external sources may need to assign different sets of encoding rules to those externally-defined model components.

7.1.5.1. Processing

ShapeChange supports a simple, yet powerful, data-flow processing model in which operations on UML models are chained starting from the input model and ending with one or more target encodings. As illustrated in Figure 22, ShapeChange can apply a number of transformations to an input model, before generating target representations for it.

ShapeChange validates the loaded Conceptual Model with respect to the requirements of UML (https://www.omg.org/spec/UML/2.5.1/About-UML/), ISO 19103:2015 (Conceptual schema language), and ISO/IEC 19507:2012 (OCL); it is thus useful as a model validation tool even in circumstances in which derivation of technology-specific implementation specifications is not intended. In such a configuration no transformers or targets are required or used.

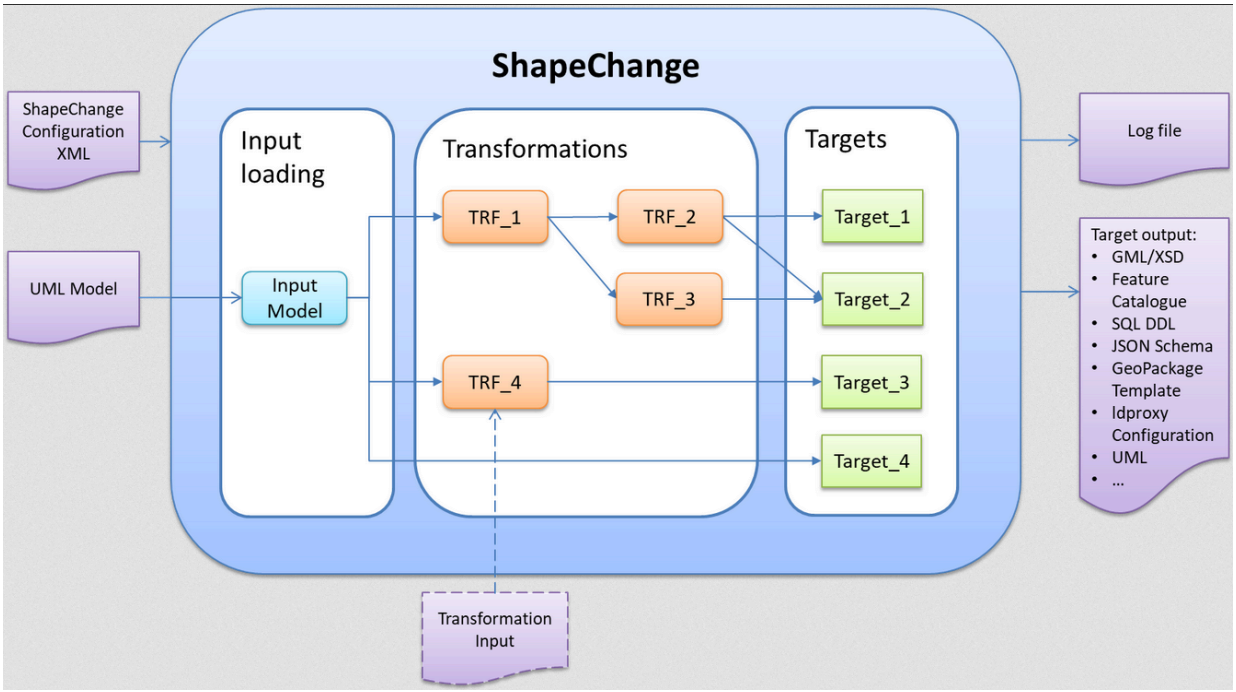


Figure 22 — ShapeChange processing model

Transformations (see: https://shapechange.net/transformations/) are applied to a given base model. This can be the input model (e.g., from an EA project file) or the result of another transformation. It is thus possible to create a chain or tree of model transformations. ShapeChange includes a wide variety of model transformations; particularly powerful is the Flattener (see: https://shapechange.net/transformations/flattener/) which may be used to simplify a model, e.g. up to the point where all complex data structures have been resolved to feature type properties whose range type is either a simple type (like Integer, Boolean or CharacterString), a code list, or an enumeration. An example use of the Flattener transformation with ArcGIS geodatabase models is documented at: https://shapechange.net/targets/arcgis-workspace/pre-processing-flattening/

Another commonly employed transformation is the Association Class Mapper (see: https://shapechange.net/transformations/association-class-mapper/) which maps a UML association class into a semantically equivalent UML class and pair of associations, as defined by ISO 19136-2:2015.

A given target configuration may be (re)used to process the output models from multiple transformers. Originally the only target of ShapeChange was the GML-based application schema, represented as a set of XML Schema documents. Currently there are many other targets that ShapeChange may be configured to produce — including non-GML XML Schema documents (e.g., the encoding of ISO 19115-1:2014 "Geographic information — Metadata — Part 1: Fundamentals"; see: https://shapechange.net/targets/xsd/iso19139/). Additional targets include, e.g., the ArcGIS Workspace, JSON Schema, Model Export (portable SCXML format), ontology (based on ISO 19150-2), SQL DDL, and UML Model (EA project file).

7.1.6. Configuration

The primary mechanism for providing arguments to ShapeChange is the project configuration file. A ShapeChange project configuration file is an XML Instance document which conforms to a custom XML Schema (https://shapechange.net/resources/schema/ShapeChangeConfiguration.xsd).

The root element of the configuration file is the `<sc:ShapeChangeConfiguration>` element. The file is then divided into various "functional elements"; key elements are:

- The `<sc:input>` element (https://shapechange.net/get-started/config/input/) defines the source of the model and some parameters controlling its interpretation.
- The `<sc:log>` element (https://shapechange.net/get-started/config/log/) defines logging parameters.
- The optional `<sc:transformers>` element (https://shapechange.net/get-started/config/transformers/) defines which transformations shall be applied to an input model, including their chaining, each in accordance with a specified set of parameters and/or rules.
- The optional `<sc:targets>` element (https://shapechange.net/targets/) specifies the configuration for each target format, e.g., XML Schema.

A project configuration file to load an EA project, create a processing log, and generate a ShapeChange XML (SCXML) file would be specified as in Figure 23:

```
<?xml version="1.0" encoding="UTF-8"?>
<ShapeChangeConfiguration
  xmlns="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
  xmlns:sc="http://www.interactive-instruments.de/ShapeChange/Configuration/1.
1"
  xmlns:xi="http://www.w3.org/2001/XInclude" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
  xsi:schemaLocation="http://www.interactive-instruments.de/ShapeChange/Configu
ration/1.1
https://shapechange.net/resources/schema/ShapeChangeConfiguration.xsd">
```

```
  <input id="INPUT">
    <parameter name="inputModelType" value="EA7" />
    <parameter name="inputFile" value="CityGML_3.0_TB17_0712.eap" />

    <parameter name="constraintLoading" value="enabled" />
    <parameter name="appSchemaNameRegex" value=".*" />

    <xi:include href="http://shapechange.net/resources/config/StandardAliases.
xml"/>
    <xi:include href="http://shapechange.net/resources/config/StandardTagAliase
s.xml"/>
  </input>

  <log>
    <parameter name="reportLevel" value="INFO" />
    <parameter name="logFile" value="generateSCXML_log.xml" />
  </log>

  <targets>
    <Target inputs="INPUT" mode=
"enabled"
class="de.interactive_instruments.ShapeChange.Target.ModelExport.ModelExport">
      <targetParameter name="outputFilename" value="CityGML_3.0_TB17_0712"/>
      <targetParameter name="schemaLocation" value="."/>
      <targetParameter name="sortedOutput" value="true"/>
      <targetParameter name="defaultEncodingRule" value="export"/>
      <xi:include href=" http://shapechange.net/resources/StandardRules.xml" />
    </Target>
  </targets>
</ShapeChangeConfiguration>
```

Figure 23 — Project configuration file to generate a ShapeChange XML (SCXML) file

The ShapeChange Model Export target is documented at: https://shapechange.net/targets/model-export/

The ShapeChange source tree includes an extensive set of unit tests (see: https://shapechange.net/get-started/examples/) that include ShapeChange configurations for loading a model, transforming it, and deriving target representations. They also include sample UML models, principally as EA project files but sometimes also as portable SCXML files.

7.1.7. Configuration resources

ShapeChange supports the use of `<xi:include>` in project configuration files, allowing for decomposition and modularization of configuration parameters. There are numerous pre-defined ShapeChange-associated resources for use in preparing ShapeChange configuration files published at: https://shapechange.net/resources/config/

The intended use of each resource file is described at: https://shapechange.net/get-started/contents/

Five of these resource files are either included by reference into typical ShapeChange project configurations, or serve as the basis for customized replacements:

- `<xi:include href="http://shapechange.net/resources/config/StandardAliases.xml"/>`: Defines mappings from domain-specific stereotypes of

packages, classes, and properties to well-known stereotypes recognized by ShapeChange. For example, mapping from «applicationSchema» to «Application Schema», and from «interface» to «Type».

NOTE 1: See: https://shapechange.net/get-started/config/input/#Stereotype_aliases

- `<xi:include href="http://shapechange.net/resources/config/StandardTagAliases.xml"/>`: Defines mappings from domain-specific tagged values of packages, classes, and properties to well-known tagged values recognized by ShapeChange. For example, from `alphaCode` to `primaryCode`, or from `characterLength` to `length`.

  NOTE 2: See: https://shapechange.net/get-started/config/input/#Tag_aliases

- `<xi:include href="http://shapechange.net/resources/config/StandardRules.xml"/>`: Defines sets of rules intended to be used together for a common purpose, e.g., a set of rules for encoding CityGML Application Domain Extensions (ADE):

```
<EncodingRule name="citygml-ade" extends="iso19136_2007">
  <rule name="req-xsd-cls-suppress-supertype"/>
  <rule name="req-xsd-cls-suppress-subtype"/>
  <rule name="req-xsd-cls-suppress-no-properties"/>
  <rule name="rule-xsd-cls-suppress"/>
  <rule name="rule-xsd-cls-adeelement"/>
  <rule name="rule-xsd-cls-mixin-classes"/>
  <rule name="rule-xsd-prop-initialValue"/>
</EncodingRule>
```

Figure 24 — Rules for encoding CityGML Application Domain Extensions (ADE) as an example of `StandardRules.xml`

NOTE 3: See https://shapechange.net/targets/xsd/#Specifying_Encoding_Rules

- `<xi:include href="http://shapechange.net/resources/config/StandardNamespaces.xml"/>`: Defines XML namespaces commonly imported by GML 3.2/3.3 application schemas, e.g., `gco`, `gml`, `xlink`, and `xsi`. See: https://shapechange.net/targets/xsd/#Namespace_Identifiers

- `<xi:include href="http://shapechange.net/resources/config/StandardMapEntries.xml"/>`: Specifies map entry files for use in target encodings; individual entries express how to encode a key (usually primitive) UML model element using a "native" encoding type. The default resource specifies a list of map entry files for GML 3.2 (and above) schema encoding:

```
<xi:include href="StandardMapEntries_iso19136_2007.xml"/>
<xi:include href="StandardMapEntries_iso19139_2007.xml"/>
<xi:include href="StandardMapEntries_sweCommon.xml"/>
<xi:include href="StandardMapEntries_iso19107.xml"/>
<xi:include href="StandardMapEntries_iso19108.xml"/>
<xi:include href="StandardMapEntries_iso19111.xml"/>
<xi:include href="StandardMapEntries_iso19115.xml"/>
<xi:include href="StandardMapEntries_iso19123.xml"/>
<xi:include href="StandardMapEntries_iso19156.xml"/>
```

```
<xi:include href="StandardMapEntries_gmlcov.xml"/>
```

Figure 25 — Listing of map entry files for GML 3.2 schema encoding

Individual map entry files in this list specify standard XML Schema implementations for many types from ISO/TC 211 and OGC Standards that are used in application schemas, e.g., `CharacterString`, `CV_RectifiedGrid`, `DateTime`, `Length`, and `TM_Position`.

Example :

```
<XsdMapEntry type="Real" xsdEncodingRules="iso19136_2007 gml33"
                         xmlPropertyType="double" xmlType="double"
                         xmlTypeType="simple" xmlTypeContent="simple"/>
```

The format and use of `<sc:XsdMapEntry>` is documented at: https://shapechange.net/targets/xsd/#XSD_Map_Entries

The resource `StandardJsonMapEntries.xml` serves a similar purpose. Its format and use is documented at: https://shapechange.net/targets/json-schema/#Map_Entries

Additional mappings, rule sets, and XML namespaces may be defined in project configurations — or added to local copies of these resource files.

7.2. Metanorma and LutaML

7.2.1. General

Metanorma is an open-source, model-based documentation system that supports the MDA approach.

It relies on LutaML, a universal information model parser, to allow navigating information models right from within the document model.

As part of the Metanorma suite, LutaML and the Metanorma-LutaML plugin have been developed iteratively to deal with the novel workflow of incorporating information model content into a target human-readable MDS document as described in this ER.

The model transformation results in a series of document elements incorporated in the MDS, including diagram rendering and tabular presentations.

7.2.2. LutaML

LutaML is an initiative grown out of Metanorma that allows parsing various machine-interpretable information models. LutaML adopts an extensible processing architecture to allow parsing different information model languages, through LutaML extensions.

Supported LutaML extensions include:

- EXPRESS, as specified in ISO 10303-11, is used heavily in smart manufacturing, Industry 4.0 use cases and in BIM, where EXPRESS itself served as the foundation of the IFC classes. The LutaML EXPRESS extension is available at: https://github.com/lutaml/lutaml-express.
- OMG UML in OMG XMI, which is the canonical format of representing UML models within XMI, an XML language defined by OMG OMG XMI 2.5. The LutaML XMI extension is available at: https://github.com/lutaml/lutaml-xmi.
- Sparx Systems Enterprise Architect XMI, the proprietary extension of Sparx Systems Enterprise Architect for the representation of UML. The LutaML Enterprise Architect-specific XMI extension is implemented within the LutaML XMI extension.
- LutaML UML, which is an ASCII syntax used to author OMG UML-compliant UML models with the possibility to be exported into OMG XMI format. The LutaML UML extension is available at: https://github.com/lutaml/lutaml-uml.

  NOTE: The LutaML UML language is documented at https://github.com/lutaml/lutaml-uml/blob/master/LUTAML.adoc

LutaML supports the dynamic referencing of elements from within a UML model. For example, individual UML classes, attributes, stereotypes, Enterprise Architect diagrams, can all be referenced through the unified interface provided by LutaML.

Collection filtering, such as to find UML classes that match certain UML stereotype, is also supported.

7.2.3. LutaML for XMI

LutaML-XMI is the LutaML extension that parses OMG XMI 2.5 into a LutaML-UML model.

Of course, each format that it reads in requires a separate plug-in to be written to process it, and the processing of different formats can be highly specialized work. That makes it important for MDA to coalesce around standard ways of expressing models as much as possible, to minimize the up-front effort of developing a new plug-in to read a new model format.

The LutaML-XMI plug-in supports parsing the proprietary XMI files generated by Sparx Systems Enterprise Architect, incorporating details only available in the vendor proprietary XML portion of the XMI file.

This plug-in has been successful in recognizing the classes it expresses, their attributes, and the relations between classes, as documented in Clause 7.2.

7.2.4. Metanorma LutaML plugin

Metanorma interfaces with the information parsed by LutaML through the Metanorma LutaML plugin (https://github.com/metanorma/metanorma-plugin-lutaml). This plugin is used to render LutaML model in human-readable formatting for MDS.

It provides a set of commands to be used within a Metanorma authoring context that invokes LutaML processing of a specified file, which generates a representation of that data usable within Metanorma.

Model navigation, dynamic referencing and collection filtering capabilities to UML models are accessible within a Metanorma document through the corresponding LutaML commands.

By default, LutaML is invoked to parse an external information model through a Metanorma AsciiDoc block command, which requires the input of the following information:

- as an argument, name of the source information model file;
- as an argument, the named context, which is the object variable name into which the data file contents are parsed, as object attributes, recursively;
- as the contents of the block, a template, in Metanorma AsciiDoc format with the Liquid template language (https://shopify.github.io/liquid/).

In effect, this provides a "meta-authoring" environment from within Metanorma. In particular, the template language allows the attributes parsed by LutaML to be incorporated in the block under the command.

7.2.5. Usage example

LutaML within Metanorma is used in the following manner.

Figure 26 shows an instance of the `[lutaml]` command in Metanorma, which instructs LutaML to process the file in `path/to/filelocation`, and pass the results of the parse into the object `package`. The body of the command then iterates through the contents of `package`, and generates Metanorma AsciiDoc using values from the variable.

```
[lutaml,path/to/filelocation,package]
--
{% for diagram in package.diagrams %}
[[figure-{{ diagram.xmi_id }}]]
.{{ diagram.name }}
image::{{ base_path }}/{{ diagram.xmi_id }}.{{ format | default: 'png' }}[]

{% if diagram.definition %}
{{ diagram.definition | html2adoc }}
{% endif %}
{% endfor %}
```

--

Figure 26 — Rendering of a UML package under LutaML

- The directives in `{% ... %}` are Liquid processing directives, including loops and conditionals.
- The variables referenced in the directives, and invoked through `{{ ... }}`, are attributes parsed by LutaML from the given source files. For example, `package.diagrams` is the list of all diagrams under the current package, and `diagram` is a loop variable containing the parsed information for one such diagram.
- The variable `diagram` contains attributes of its own which LutaML has parsed; the XMI ID attribute for the diagram,
  - `{{ diagram.xmi_id }}` is used in conjunction with the LutaML parameter `{{ image_base_path }}` in order to define the file location of the associated image file.
  - `{{ diagram.xmi_id }}` is also used with the prefix `figure-` to define the anchor for the image (`[[...]]`), to be used in cross-references.
  - The markup `.{{ diagram.name }}` is used to insert the name attribute of the diagram as the image caption.

While the `[lutaml]` command can be used individually to build up an MDS, its use would be highly repetitive, as an MDS will render each UML class and package it is applied to in the same way.

For that reason, Metanorma defines the command `[lutaml_uml_datamodel_description]` to iterate through a sequence of UML packages, rendering each in a consistent way. The template that would be used for each class is predefined, and users do not have to supply their own Liquid template text.

The general format for the `[lutaml_uml_datamodel_description]` command is given in Figure 27. This command generates a Metanorma representation of the UML class diagram contained in the XMI file `path/to/example.xmi`.

```
[lutaml_uml_datamodel_description, path/to/example.xmi,config.yaml]
--
[.before]
....
my text
....

[.diagram_include_block, base_path="requirements/", format="emf"]
....
Diagram text
....

[.include_block, package="Another", base_path="spec/fixtures"]
....
my text
```

```
....

[.include_block, base_path="spec/fixtures"]
....
my text
....

[.before, package="Another"]
....
text before Another package
....

[.after, package="Another"]
....
text after Another package
....

[.after, package="CityGML"]
....
text after CityGML package
....

[.after]
....
footer text
....
--
```

Figure 27 — `lutaml_uml_datamodel_description` command

Within the `lutaml_uml_datamodel_description` command, there is a placeholder for UML content, `[.diagram_include_block]` which automatically invokes Figure 26, the predefined Liquid template specifying how UML information is included for each package within the command.

The `[.diagram_include_block]` block includes:

- `base_path`, a required attribute for path prefixes to supply for diagram image;
- `format`, an optional attribute that tells what file extension to use when including diagram a file.

The remaining blocks specified within the command give text to interpolate:

- before or after each package in the loop (`[.before, package="Another"]`, `[.after, package="CityGML"]`); or
- before or after all packages have been iterated through (`[.before]`, `[.after]`).

There is also provision for text to be interpolated in predefined positions within each package (`[.package_text, position="after", package="Another"]`).

NOTE 1: Only `before` and `after` are currently defined as values for `position`.

Last but not least, there is provision for one or more files matching a base path specification to be invoked for inclusion in the command (`[.include_block, position="before", base_path="requirements/requirements_class_"]`).

The `config.yaml` parameter of the command is optional. The nominated YAML file specifies which packages to process in the command, in which order; rendering style instructions; and the location of the root package. An example of the YAML file is provided at Figure 28.

```
---
packages:
  # includes these packages
  - "Package *"
  - two*
  - three
  # skips these packages
  - skip: four
render_style: data_dictionary
section_depth: 2
```

Figure 28 — YAML configuration for `lutaml_uml_datamodel_description` command

The example configuration in Figure 28 indicates that packages matching the regexp `Package` , the regexp `two`, and the name `three` are to be processed, in that order, while `four` is to be skipped.

It indicates the rendering style, which can be one of `entity_list`, `data_dictionary`, or `default`.

Finally, it indicates the limit if any to how deep the recursive iteration of packages can go.

NOTE 2: The styles `entity_list` and `data_dictionary` were developed specifically for CityGML, and the different level of detail it gives for Clause 7 and Clause 8. If finer differentiations between rendering formats are required, the configuration of rendering styles will need to be made more granular.

8

MODEL SPECIFICATION ISSUES AND RECOMMENDATIONS

8 MODEL SPECIFICATION ISSUES AND RECOMMENDATIONS

8.1. General

The D144 subtask worked directly on these conceptual models:

- CityGML 3.0 Conceptual Model, published by the CityGML SWG as XMI generated from an EAP file
- DGGS Conceptual Model, published by ISO/TC 211 HMMG as an EAP file

During the development of the D144 prototypes, a number of modelling issues were discovered and documented in this clause.

8.2. CityGML 3.0 conceptual model

8.2.1. Annotation format in model editing tool

Clause 4.14 noted the challenge in coordinating supplementary truth in a standards document, authored in an environment like Metanorma, against annotations included in the derived truth in a standards document, and authored in the model authoring tool.

Sparx Systems Enterprise Architect is the modelling tool used for the CityGML 3.0 conceptual model, and allows for the annotation of UML objects as description, notes, using a rich-text editor. Sparx Systems Enterprise Architect encodes these rich text boxes with either the Microsoft RTF format or in HTML.

The challenges of including such content inside an MDS include:

- The content of these fields is generally inconsistent. Some fields are encoded in HTML, some in RTF.
- Some authors embed lightweight markup languages such as AsciiDoc or Markdown syntax, but encoded in HTML or RTF format instead of plain text.

The annotations in CityGML reflected a drive to format text in annotations without coordination or planning.

For the downstream processing of annotations, it is important to both identify the formatting scheme used for annotations within the model, and then convert it into the same markup scheme as that used in the target document.

Given that Metanorma AsciiDoc is the official markup language used by OGC for its standards documents, all annotations authored within model authoring tools must be marked up using the same markup language in order to allow MDS generation.

RECOMMENDATION 1
/rec/mbs/annotation-format
Information models intended to be used in model-driven standards should have annotations in a standardized format for the consumption of model-driven authoring tools. OGC officially uses Metanorma for the standards authoring, therefore all annotations authored within model authoring tools should be written in a markup language supported by Metanorma. Annotations written in languages that Metanorma cannot automatically detect should include in its content an explicit indication of the markup language used.

8.2.2. Presentation styles of UML model-based standards

8.2.2.1. General

The OGC CityGML 3.0 conceptual model document used two separate styles for presenting UML objects for the ease of use for the reader. This was realized by a different configuration of the Metanorma call to LutaML as it iterated over the conceptual model for each style, resulting in a different set and arrangement of parsed model attributes being passed on to Metanorma.

RECOMMENDATION 2
/rec/mbs/model-presentation-style
The presentation style of information models, such as UML models, within an MDS, should be standardized for consistency. There could one default style with potentially a few more styles to choose for visual representation of different model types, such as UML package, UML class, UML association. Ad-hoc presentation styles intended to display core information on information models should be discouraged as they can confuse the reader. Such presentation styles instead can be useful to show additional information about those models.

8.2.2.2. Entity list style

In the entity list style, each class and data type in a package is presented in tables, with one row per class and data type, giving an identifier for the entity and a paragraph description. No further detail of the classes was provided.

```
7.6.1. Requirements
7.6.2. Class definitions
7.6.3. Additional Information
```

Figure 29 — Order of subsections per UML package in the entity list style

A sample of the “Class definitions” table is shown in Figure 30.

7.6.2. Class definitions

Table 25 — Classes defined in Dynamizer (ApplicationSchema)

NAME	DESCRIPTION
AbstractAtomicTimeseries «FeatureType»	AbstractAtomicTimeseries represents the attributes and relationships that are common to all kinds of atomic timeseries (GenericTimeseries, TabulatedFileTimeseries, StandardFileTimeseries). An atomic timeseries represents time-varying data of a specific data type for a single contiguous time interval.
AbstractTimeseries «FeatureType»	AbstractTimeseries is the abstract superclass representing any type of timeseries data.
CompositeTimeseries «FeatureType»	A CompositeTimeseries is a (possibly recursive) aggregation of atomic and composite timeseries. The components of a composite timeseries must have non-overlapping time intervals.

Figure 30 — Sample "Class definitions" table from 20-010, 7.6.2

8.2.2.3. Data dictionary style

In the data dictionary style, each class and data type in a package is presented in a separate subclause, with tables presenting not only the identifier and a paragraph description, but also superclasses, stereotypes, and in the case of classes, all class attributes with their cardinalities and (hyperlinked) types. Any applicable basic types, unions, and code lists are also enumerated.

```
8.6. Dynamizer
8.6.1. Classes
8.6.2. Basic types
8.6.3. Unions
8.6.4. Code lists
8.6.5. Data types
8.6.6. Enumerations
```

Figure 31 — Order of subsections per UML package in the data dictionary style

The first section extracted from the PIM is the “metadata”. A sample is shown at Figure 32.

8.3. Appearance

Table 219 — Metadata of Appearance (ApplicationSchema)

DESCRIPTION:	The Appearance module supports the modelling of the observable surface properties of City GML features in the form of textures and material.
PARENT PACKAGE:	CityGML
STEREOTYPE:	«ApplicationSchema»

Figure 32 — Sample "metadata" block, 20-010, 8.3

The second section extracted provides the classes. LutaML will iterate through all classes in the UML package generating one table each. A sample is shown at Figure 33.

8.3.2. Basic types

8.3.2.1. Color

Table 237 — Metadata of Color (BasicType)

DEFINITION:	Color is a list of three double values between 0 and 1 defining an RGB color value.
SUBCLASS OF:	DoubleBetween0and1List
STEREOTYPE:	«BasicType»
CONSTRAINT:	`lengthOfList:` `inv: list->size() = 3`

Figure 33 — Sample "metadata" block, 20-010, 8.3.2

The third section extracted provides the “Basic types” (stereotype `BasicType`). LutaML will iterate through all basic types in the UML package generating one table each. A sample is shown at Figure 34.

8.6.1. Classes

8.6.1.1. AbstractAtomicTimeseries

Table 273 — Metadata of AbstractAtomicTimeseries (FeatureType)

DEFINITION:	AbstractAtomicTimeseries represents the attributes and relationships that are common to all kinds of atomic timeseries (GenericTimeseries, TabulatedFileTimeseries, StandardFile Timeseries). An atomic timeseries represents time-varying data of a specific data type for a single contiguous time interval.
SUBCLASS OF:	AbstractTimeseries
STEREOTYPE:	«FeatureType»

Table 274 — Attributes of AbstractAtomicTimeseries (FeatureType)

ATTRIBUTE	VALUE TYPE AND MULTIPLICITY	DEFINITION
adeOfAbstract Atomic Timeseries	ADEOfAbstractAtomic Timeseries [0..*]	Augments AbstractAtomicTimeseries with properties defined in an ADE.
observation Property	CharacterString [1..1]	Specifies the phenomenon for which the atomic timeseries provides observation values.
uom	CharacterString [0..1]	Specifies the unit of measurement of the observation values.

NOTE: Unless otherwise specified, all attributes and role names have the stereotype «Property».

Figure 34 — Sample "Basic types" block, 20-010, 8.6.1

The next section extracted provides the unions. The CityGML 3.0 document does not define Unions.

Then the code lists are extracted. Similar to above, LutaML will iterate through all code lists in the UML package generating one table each. A sample is shown at Figure 35.

8.6.4. Code lists

8.6.4.1. AuthenticationTypeValue

Table 290 — Metadata of AuthenticationTypeValue (CodeList)

DEFINITION:	AuthenticationTypeValue is a code list used to specify the authentication method to be used to access the referenced sensor service. Each value provides enough information such that a software application could determine the required access credentials.
SUBCLASS OF:	None
STEREOTYPE:	«CodeList»

Figure 35 — Sample "Code lists" block, 20-010, 8.6.4

Second to last the “data types” are extracted. LutaML will iterate through all data types in the UML package generating one table each. A sample is shown at Figure 36.

8.6.5. Data types

8.6.5.1. ADEOfAbstractAtomicTimeseries

Table 294 — Metadata of ADEOfAbstractAtomicTimeseries (DataType)

DEFINITION:	ADEOfAbstractAtomicTimeseries acts as a hook to define properties within an ADE that are to be added to AbstractAtomicTimeseries.
SUBCLASS OF:	None
STEREOTYPE:	«DataType»

Figure 36 — Sample "Data types" block, 20-010, 8.6.5

The last section extracted provides the enumerations. LutaML will iterate through all enumeration objects in the UML package generating one table each. A sample is shown at Figure 37.

8.6.6. Enumerations

8.6.6.1. TimeseriesTypeValue

Table 309 — Metadata of TimeseriesTypeValue (Enumeration)

DEFINITION:	TimeseriesTypeValue enumerates the possible value types for GenericTimeseries and TimeValuePair.
STEREOTYPE:	«Enumeration»

Table 310 — Values of TimeseriesTypeValue (Enumeration)

LITERAL VALUE	DEFINITION
int	Indicates that the values of the GenericTimeseries are of type “Integer”.
double	Indicates that the values of the GenericTimeseries are of type “Double”.

Figure 37 — Sample "Enumerations" block, 20-010, 8.6.6

8.2.3. Multiplicity notation

The CityGML 3.0 Conceptual Model employs an uncommon UML notation for specifying the multiplicity of an association role (technically the `MultiplicityElement` of an `Association` end).

In Figure 38, the notations `*` and `1` (red dashed boxes) are alternatives to the complete specifications of `0..*` and `1..1`, respectively. In comparison `0..1` (green dashed box) is a complete specification.

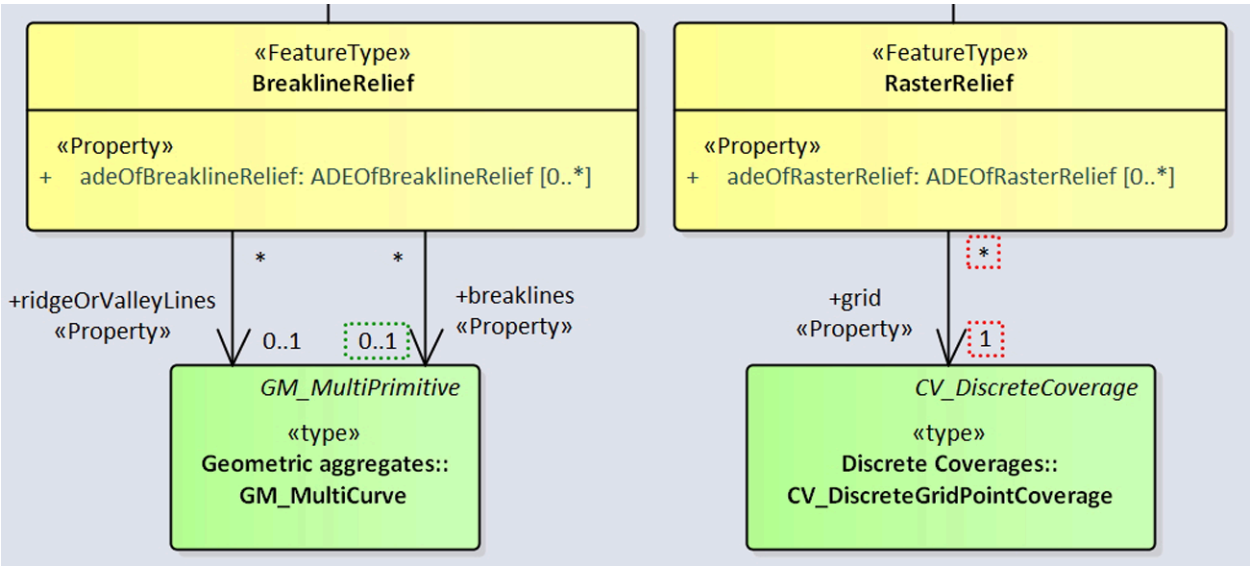


Figure 38 — CityGML 3.0 Conceptual Model Multiplicity Notation

This alternative notation is used inconsistently throughout the standard (both the document and EAP) — in particular it is the case that `*` is used with association roles while `0..*` is used with attributes. The alternative notation and its use are not documented in the standard itself and may be either confusing or ambiguous to some users of the standard. It is also inconsistent with the requirements of OMG UML 2.5.

OMG Unified Modeling Language (UML) (https://www.omg.org/spec/UML/2.5.1/About-UML/) specifies the notation for "Multiplicity Elements" in Clause 7.5.4.1 as follows:

> The multiplicity bounds may be shown in the format:
>
> `<lower-bound> .. <upper-bound>`
>
> where `<lower-bound>` is a ValueSpecification of type `Integer` and `<upper-bound>` is a `ValueSpecification` of type `UnlimitedNatural`. The star character (`*`) is used as part of a multiplicity specification to represent an unlimited upper bound.
>
> If the multiplicity is associated with a `MultiplicityElement` whose notation is a text string (such as an attribute), the multiplicity string is placed within square brackets (`[ ]`) as part of that text string.
>
> If the multiplicity is associated with a `MultiplicityElement` that appears as a symbol (such as an `Association` end), the multiplicity string is displayed without square brackets and may be placed near the symbol for the element.
>
> If the lower bound is equal to the upper bound, then an alternate notation is to use a string containing just the upper bound. For example, `1` is semantically equivalent to `1..1` multiplicity. A multiplicity with zero as the lower bound and an unspecified upper bound may use the alternative notation containing a single star `*` instead of `0..*` multiplicity.

An example of the use of alternative notation in class diagrams as presented in OMG UML 2.5, Figure 8.2 is given in Figure 39; note that the use of alternative notation is consistent for all properties in the diagram.

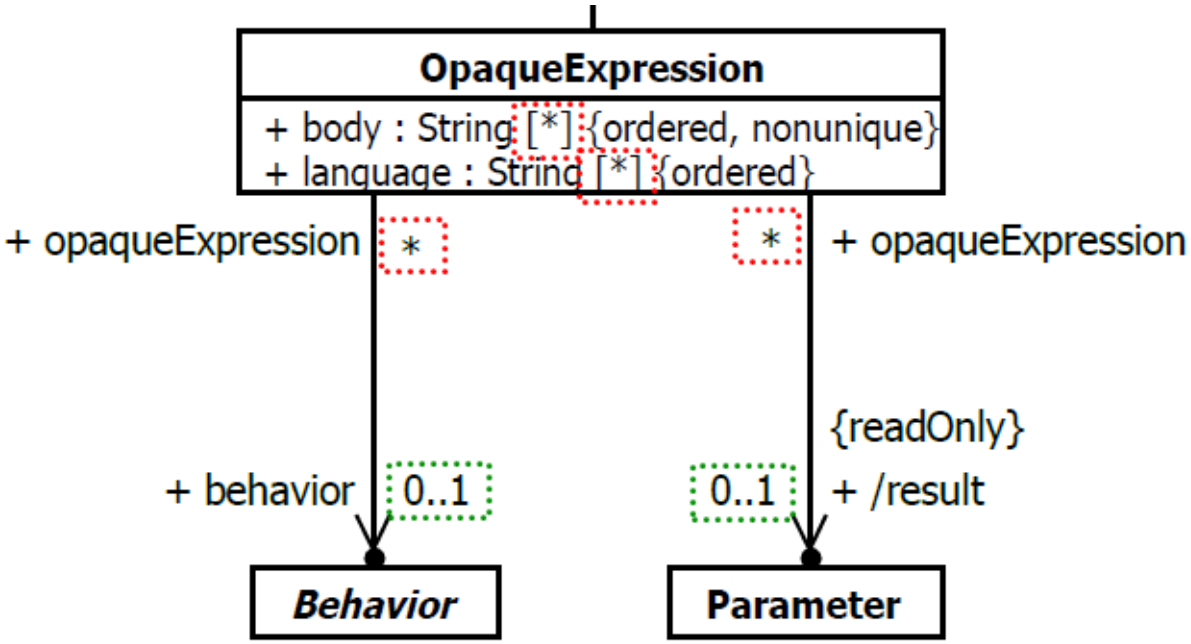


Figure 39 — UML 2.5.1 Multiplicity Element Notation Examples

ISO 19103:2015, Clause G.8.2 (Informative) states:

> The multiplicity of an association-end can be one of exactly-one (`1`), zero-or-one (`0..1`), one-or-more (`1..`), zero-or-more (`0..`), an interval (`n..m`) or a set of given numbers (`m, n, o, p`).

ISO 19103:2015, Clause H.13 “Former assumptions”, states:

> In earlier models a lack of explicit multiplicity on association ends may have been interpreted as being fixed to `[1]` or `[0..*]`. This is no longer allowed, and explicit multiplicity must be provided.

Although ISO 19103:2015 does not mention the use of alternative notation, it does sanction the use of `1` rather than `1..1`.

Use of the alternative notation in the case of `0..` may be unclear to the editors and users of conceptual models, particularly as this alternative notation is not identified in ISO 19103:2015 as suitable for use in ISO/TC 211 developed standards. For example, ISO 19170-1:2021 consistently uses `1` and `0..`, as appears to be the case for most, and possibly all, other HMMG conceptual schemas.

The CityGML 3.0 Conceptual Model potentially further confuses matters by using the standard notation for attribute multiplicity but the alternative notation for association role multiplicity.

RECOMMENDATION 3

/rec/mbs/uml/multiplicity

Adopt a consistent multiplicity notation for all UML properties, whether expressed in the form of a document or a class diagram. In particular adopt the pattern used in ISO/TC 211:

1. If the lower bound is equal to the upper bound, then use a string containing just the upper bound.
2. If the lower bound is not equal to the upper bound, then use a string containing both bounds. A multiplicity with zero as the lower bound and an unspecified upper bound is expressed as `0..*`.

Document the multiplicity notation in the standard and then employ it consistently throughout the model.

8.2.4. Conceptual model dependencies

OGC 20-010, Clause 8.2 documents requirements class http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core wherein dependencies are asserted with respect to six external standards: ISO 19103:2015, ISO 19107:2003, ISO 19109:2015, ISO 19111:2019, ISO 19123:2005, and OASIS xAL v3.0 (OASIS CIQ v3.0). As documented in the CityGML 3.0 Conceptual Model EAP these direct dependencies are illustrated in Figure 40.

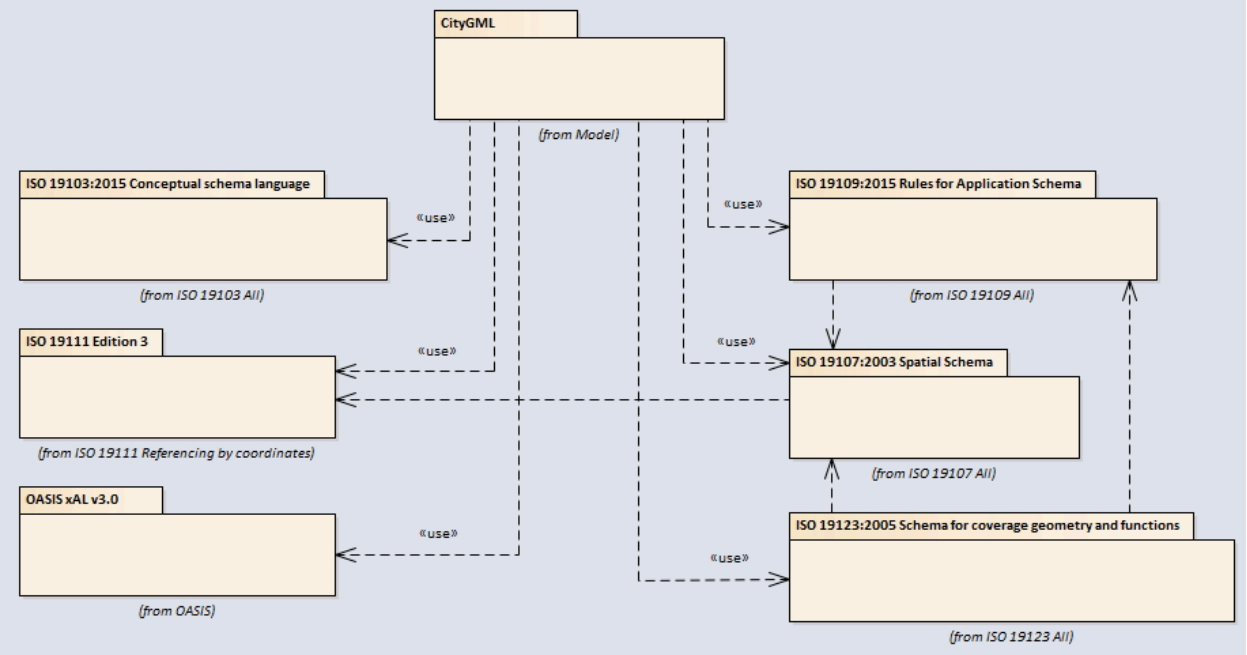


Figure 40 — CityGML 3.0 Part 1 — Conceptual Model UML Package Primary Dependencies

Requirements class http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core is incorrect when stating a dependency only on ISO 19111:2019. There is a dependency on the «`type`» `SC_CRS` class, which is no longer present (and has no direct correspondence) in ISO 19111:2019; in this case the dependency is on ISO 19111:2007. However «`FeatureType`»

`CityModel`, attribute engineeringCRS specifies a range of "EngineeringCRS", which is not a class in ISO 19111:2007; it is only present in Edition 3. As class "EngineeringCRS" has a direct correspondence to the «type» `SC_EngineeringCRS` class in ISO 19111:2007 consistent adoption of a dependency on, and classes from, ISO 19111:2007 is the most appropriate resolution to these inconsistencies.

Requirements class http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core is incorrect when failing to identify a dependency on ISO 19108:2002. CityGML 3.0 Table 5 and Section 9.1.14 specify the use of the «`Union`» `TM_Position` class which in turn depends on the use of `TM_TemporalPosition` with many subclasses and related types. «`Union`» `TM_Position` is the range of attributes `startTime` and `endTime` of the «`FeatureType`» `Dynamizer` class.

Requirements class http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core states a dependency on ISO 19103:2015; e.g., the range of attribute value of the «`DataType`» `DateAttribute` class in the CityGML 3.0 Conceptual Model is specified as «`interface`» `Date` from ISO 19103:2015. There are, however, indirect dependencies on ISO/TS 19103:2005 (the earlier edition as a Technical Specification) through the other standards dependencies. Mixing dependencies on different editions of the same standard may result in unexpected outcomes in the integrated conceptual model. It also adds complexity in the derivation of platform-specific models where distinct authoritative target implementations may exist for each edition of the external standard (as is typically the case for XML Schema), thus complicating (and possibly precluding) the use of a "mix and match" technique when deriving an Implementation Specification for that technology target.

As the ISO/TC 211 Harmonized Model conceptual schemas include all editions of each standard developed by ISO/TC 211 within a single EAP file, it is helpful to trim the assemblage by eliminating all but one edition of each standard upon which a dependency is to be asserted.

Eliminating excessive content from the EAP improves performance while ensuring that there is no possibility of confusion among same-named classes or inadvertently using a class that does not exist in the intended standard (or version of that standard).

In the present analysis, ISO/TS 19103:2005 was adopted (rather than ISO 19103:2015) in order to ensure consistency in both direct and indirect dependencies.

RECOMMENDATION 4
/rec/mbs/model-dependencies
When preparing a PIM for PIM-to-PSM, all models that the PIM depends on should be made available within the model package. All unused information models should be trimmed away from the PIM used for model conversation.

RECOMMENDATION 5
/rec/mbs/model-dependencies
When preparing a PIM for PIM-to-PSM, all models that the PIM depends on should be made available within the model package. All unused information models should be trimmed away from the PIM used for model conversation.

RECOMMENDATION 5

In the case of requiring packages from the ISO/TC 211 HMM, always reduce the included standard packages from the HMM to exactly those required (or copy just the required packages into a new EA project); this avoids potential confusion between property range types defined either in different standards or in different versions of the same standard. It also makes clear (during development) when a withdrawn ISO standard is being used in lieu of its modern replacement.

8.2.5. Conceptual Model Completeness

OGC 20-010, Clause 8.2 documents requirements class http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core which states a dependency on OASIS xAL v3.0 (OASIS CIQ v3.0). In particular, the range of association role xalAddress of the «FeatureType» Address class in the CityGML 3.0 Conceptual Model is specified as a property-free «DataType» XALAddress, as shown in Figure 41.

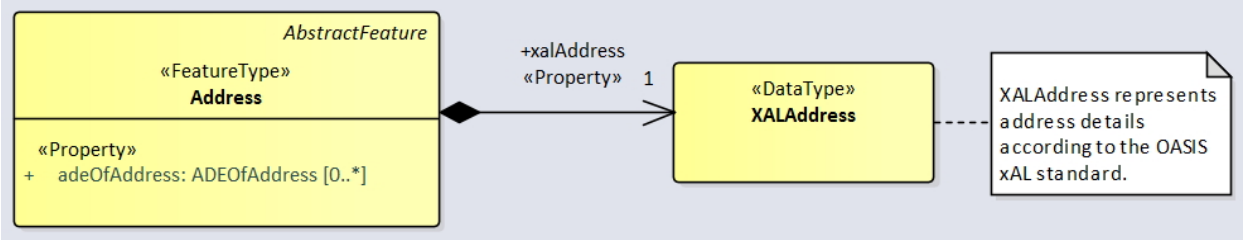


Figure 41 — CityGML 3.0 Part 1 — «FeatureType» Address

According to its textual definition, XALAddress is intended to operate as a surrogate for "address details" from the OASIS xAL standard. The "OASIS xAL v3.0" UML package in the CityGML 3.0 Conceptual Model is empty. In consequence it is not possible to derive any complete platform-specific models from the CityGML 3.0 Conceptual Model. While in the case of XML Schema there exists an authoritative Implementation Specification (http://docs.oasis-open.org/ciq/v3.0/cs02/xsd/default/xsd/xAL.xsd), there is no corresponding authoritative Implementation Specification for other technologies, e.g. JSON Schema.

RECOMMENDATION 6

/rec/mbs/model-completeness

A complete conceptual model should be defined in order to be able to derive complete Implementation Specifications for arbitrary (including future) target technologies. While it is possible to take a short-cut if only one target-specific Implementation Specification is intended (dependent on the availability of existing target-specific encodings for classes and submodels that are not completely defined in the conceptual model), that approach limits the utility of the conceptual model in areas in which it is incomplete.

Preparation of a derived target-specific Implementation Specification may be dependent on target-related conceptual models/schemas. For example, in the case of implementations based on JSON, the well-known GeoJSON format for encoding geographic data structures may

be employed. While there is published an overall JSON-based schema (https://geojson.org/schema/GeoJSON.json) as well as GeoJSON type-specific schemas (e.g., https://geojson.org/schema/Point.json), there exists no corresponding conceptual model that might be used in either documenting an Implementation Specification model in UML, or as an adjunct to the PIM for use in deriving alternative technology-specific Implementation Specifications.

RECOMMENDATION 7
/rec/mbs/model-reference
To aid in the development of OGC Standards incorporating conceptual models that are intended for use in deriving Implementation Standards it is desirable to create reference (but likely non-authoritative) UML-based conceptual models where such do not already exist. This includes both cases where no UML-based conceptual model exists (e.g., OASIS xAL and GeoJSON), and cases where a UML-based conceptual model exists "on paper" but is not readily available in XMI (or as a Sparx Systems Enterprise Architect project), e.g., SWE Common. In some cases, it may be sufficient to only create partial reference UML-based conceptual models that include only those classes and submodels relevant to OGC standards (e.g., only spatial or temporal representations).

8.2.6. External Conceptual Model Readiness

Developing a technology-specific Implementation Standard based on the CityGML 3.0 Conceptual Model requires determining encodings corresponding to UML classes defined in external conceptual models. When there exist suitable authoritative encodings then UML classes in the external conceptual models can be mapped to individual corresponding encoding representations in XML Schema. As it happens, all external conceptual models upon which the CityGML 3.0 Conceptual Model is dependent have authoritative XML Schemas — published by either ISO or OASIS. This is not the case for other technology-specific encodings, e.g., JSON Schema.

When there do not exist suitable authoritative encodings, it becomes necessary to develop those encodings in addition to developing the CityGML 3.0 Conceptual Model encoding itself. In such circumstances it is necessary to assess the readiness of those external conceptual models for use alongside of the CityGML 3.0 Conceptual Model itself.

Ignoring "basic" types defined in ISO 19103:2015, the (revised) CityGML 3.0 Conceptual Model includes direct dependencies on 19 classes defined in external conceptual models, as follows:

- ISO 19107:2019
  - `GM_Solid`
  - `GM_Point`
  - `GM_MultiSurface`
  - `GM_MultiCurve`
  - `GM_Surface`
  - `GM_MultiPoint`
  - `GM_TriangulatedSurface`
  - `GM_Object`
  - `DirectPosition`
- ISO 19108:2002
  - `TM_Position`
  - `TM_Duration`
- ISO 19109:2015
  - `AnyFeature`
- ISO 19111:2019
  - `SC_CRS`
  - `SC_EngineeringCRS`
- ISO 19123:2005
  - `CV_DiscreteGridPointCoverage`
- ISO 19136-1:2020 (introduced by the model revisions)
  - `doubleList`
  - `Code`
  - `MeasureOrNilReasonList`

- OASIS xAL (OASIS CIQ v3.0)
  - `XALAddress`

Almost all of these classes are complex and have dependencies in turn on other classes. In some cases, those dependencies draw from classes or submodels defined in additional standards. There are five cases of potential dependencies on conceptual models for metadata, as follows:

- ISO 19115:2003 or ISO 19115-1:2014
  - `CI_Citation`
  - `EX_Extent`
  - `RS_Identifier`
  - `LanguageCode`
- ISO 19115:2003 or ISO 19157:2013
  - `DQ_PositionalAccuracy`

Presumptively these dependencies do not actually exist with respect to the intent of the CityGML 3.0 Conceptual Model; these cases from ISO 19115:2003 were ignored, leaving six conceptual schemas published by the ISO/TC 211 Harmonized Model Maintenance Group (HMMG: https://github.com/ISO-TC211/HMMG). These conceptual schemas were evaluated for their ability to support MDA-facilitated application in derived Implementation Standards.

Excluded from the review and revision was one package — ISO 19109:2015 “Rules for Application Schemas::Examples”. This package contains many informative uses of classes that are not intended for use beyond graphic illustration.

Thirty-five cases of the deprecated syntactic construct `Sequence<...>` and three cases of `Set<...>` were identified across all six standards. These 38 occurrences were individually revised in accordance with guidance given in ISO 19103:2015, Table 8, as in Figure 42:

Table 8 — Keywords for combinations of ordering and uniqueness

Keyword	Combination
set	unordered, unique elements (default)
bag	unordered, nonunique elements
orderedSet	ordered, unique elements
list (or sequence)	ordered, nonunique elements

EXAMPLE 1 position : DirectPosition [1..*] {set} instead of position : Set<DirectPosition>

EXAMPLE 2 coordinate : Number [0..*] {sequence} instead of coordinate : Sequence <Number>

This mechanism can replace the collection templates defined in the previous version of this International Standard. The collection templates are retained for direct backward compatibility and described in 7.3.2 – 7.3.5.

Figure 42 — ISO 19103:2015 Table 8

Three cases of legacy class names containing pairs of angle-brackets were revised, as follows:

- Class `TransfiniteSet<DirectPosition>` was renamed to `TransfiniteSetOfDirectPositions` consistent with ISO 19107:2019.
- Class `Sequence<Character>` was renamed to `SequenceOfCharacters`, although this class no longer exists in ISO 19103:2015.
- Class `Dictionary<MemberName, TypeName>` was deleted as it served only illustrative purposes and no longer exists in ISO 19103:2015.

Four cases were revised in which an Aggregation association was used with a «`Union`» datatype range rather than the proper use of a Composition, as follows:

- Property coordinateSystem of class SC_ImageCRS with range CS_ImageCS (ISO 19111:2019)
- Property coordinateSystem of class SC_EngineeringCRS with range CS_EngineeringCS (ISO 19111:2019)
- Property coordinateSystem of class SC_GeodeticCRS with range CS_GeodeticCS (ISO 19111:2019)
- Property valueComponent of class CompositeValue with range Value (ISO 19136:2007)

Class CV_CommonPointRule was defined twice in the ISO 19123:2005 conceptual schema, appearing identically in packages “Coverage Core” and “Segmented Curve”; it was deleted from the “Segmented Curve” package.

Twelve classes with more than one supertype were identified (e.g., TM_TemporalCRS); multiple inheritance requires special consideration when developing technology-specific model implementations.

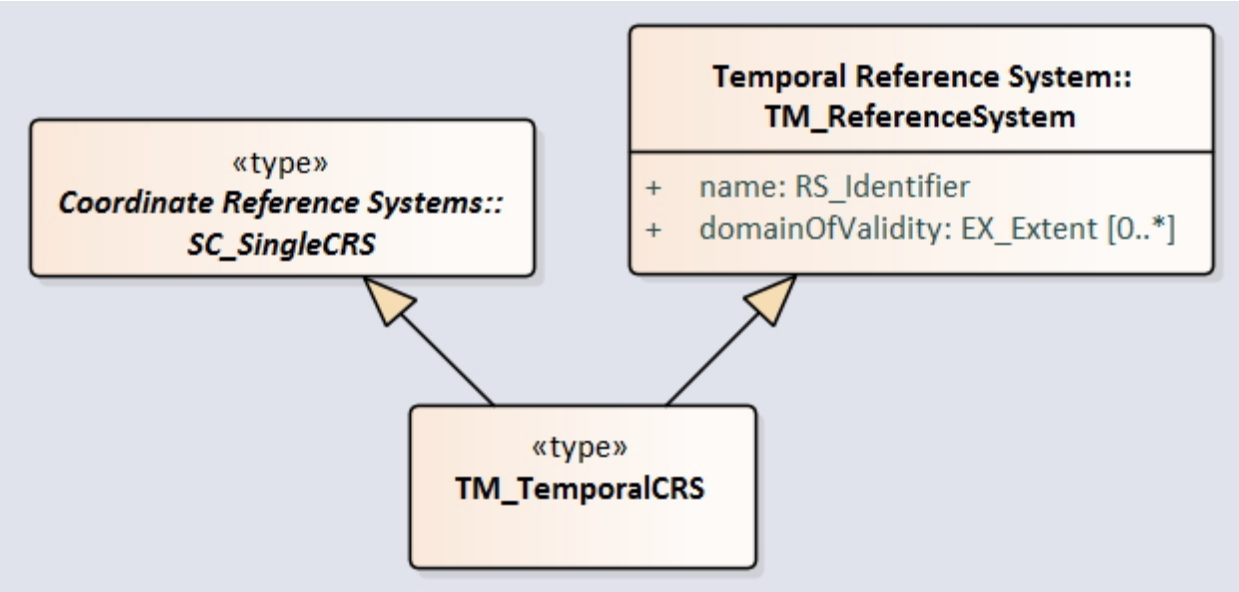


Figure 43 — TM_TemporalCRS with more than one supertype

Neither the consistent use of package, class, and property stereotypes, nor the application of tag definitions and populated values were assessed. However the significant tagged value xsdEncodingRule was found to be irregularly populated with the value `iso19136_2007_INSPIRE_Extensions`. No attempt was made to remove these values, instead actions were taken to block their use during model processing.

RECOMMENDATION 8
/rec/mbs/model-mda-ready
To aid in the development of OGC Standards incorporating conceptual models from other standards that are intended for use in deriving Implementation Standards it is desirable to establish “MDA-ready” versions of those conceptual modes for reuse by OGC model developers and standards editors. Readiness may consist of a combination of judicious revision (e.g., adoption of modern syntax that does not affect semantics) and documentation of potential hazards (e.g., use of multiple inheritance, unusual stereotypes, and tag definitions with values).

8.2.7. Novel stereotypes

The CityGML 3.0 Conceptual Model includes twenty (20) classes stereotyped «`TopLevelFeatureType`», defined as “denotes features that represent the main components of the conceptual model” (including, e.g., `CityObjectGroup`, `GenericLogicalSpace`, `GenericOccupiedSpace`, and `GenericThematicSurface`).

As an example of the typical use of this stereotype Figure 44 illustrates the relationships between «TopLevelFeatureType» ReliefFeature, «FeatureType» AbstractReliefComponent, and their shared generalization «FeatureType» AbstractSpaceBoundary in the EAP.

ISO 19109:2015, when defining the General Feature Model (GFM, the basis for application schemas like CityGML), states "InheritanceRelation is the class for a relationship between a more general feature type (supertype) and one specialized feature type (subtype)" (formalized as /req/general/inheritance). The red dotted line indicates the class to which an incorrect stereotype has been applied; the InheritanceRelation between it and «FeatureType» AbstractSpaceBoundary fails to satisfy /req/general/inheritance of ISO 19109:2015.

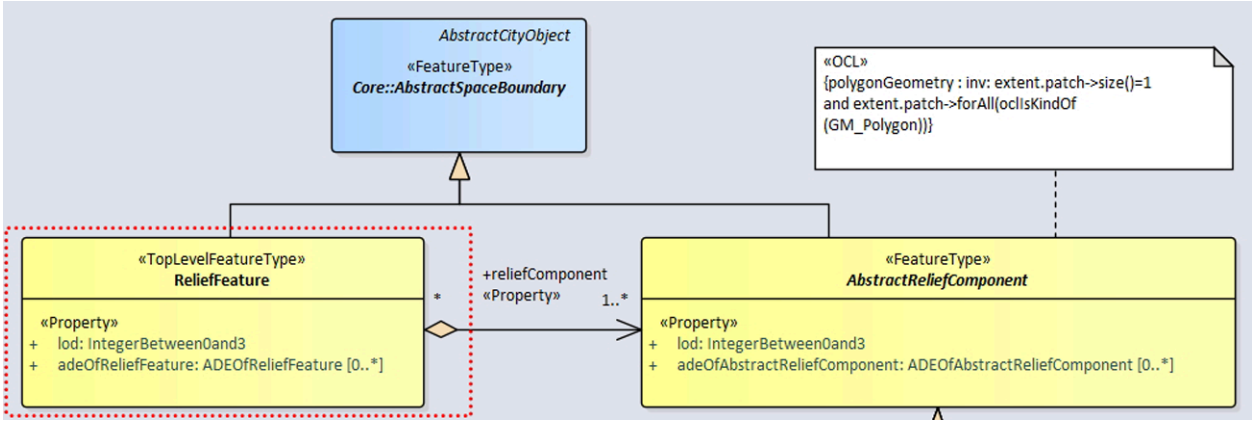


Figure 44 — CityGML 3.0 Part 1 — Example of existing class stereotypes

Figure 45 illustrates the class (inside of the blue dotted line) that was revised to add the missing «FeatureType» stereotype. In the revised model each such class has two applied stereotypes; <<FeatureType>> is consistent with that of the superclass (GFM requirement), «TopLevelFeatureType» serves as a semantics-carrying indication specific to the CityGML 3.0 Conceptual Model ("features that represent the main components").

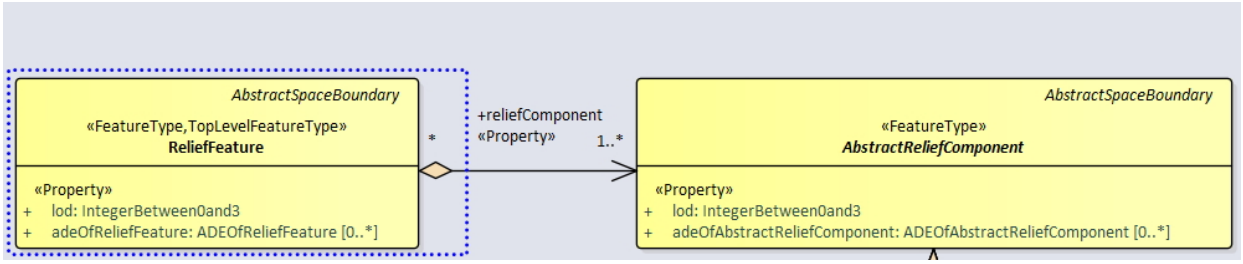


Figure 45 — CityGML 3.0 Part 1 — Example of revised class stereotypes

RECOMMENDATION 9
/rec/mbs/conceptual-conformance
Conceptual models of Application Schemas should comply with all requirements of ISO 19109: 2015.

The CityGML 3.0 Conceptual Model includes four classes stereotyped «ObjectType», defined as "represents objects that have an identity, but are not features"; these classes are: ImplicitGeometry, CityObjectRelation, TextureAssociation, and Role. In effect this stereotype denotes CityGML 3.0 Conceptual Model classes that are «Type» but not «FeatureType», the stereotype «Type» (or «type») being specifically reserved for classes defined by external standards. Although stereotype «ObjectType» serves as a useful semantics-carrying indication specific to the CityGML 3.0 Conceptual Model it has no specific implications for Implementation Specifications beyond those of stereotype «Type» (or «Interface»).

8.2.8. Reuse of Conceptual Schemas

OGC 20-010, Table 11 defines «BasicType» Code as "a basic type for a String-based term, keyword, or name that can additionally have a code space." This is not, in fact, what is modeled in the EAP. In Figure 46 the left-side submodel (inside of the red dotted line) is that in the EAP. Both «interface» classes are from ISO 19103:2015.

Note that while the concept of "additionally having a code space" is modeled, there is no capability to carry the code (string) itself — class «BasicType» Code is not a subclass of «interface» CharacterString (although «interface» URI is such a subclass).

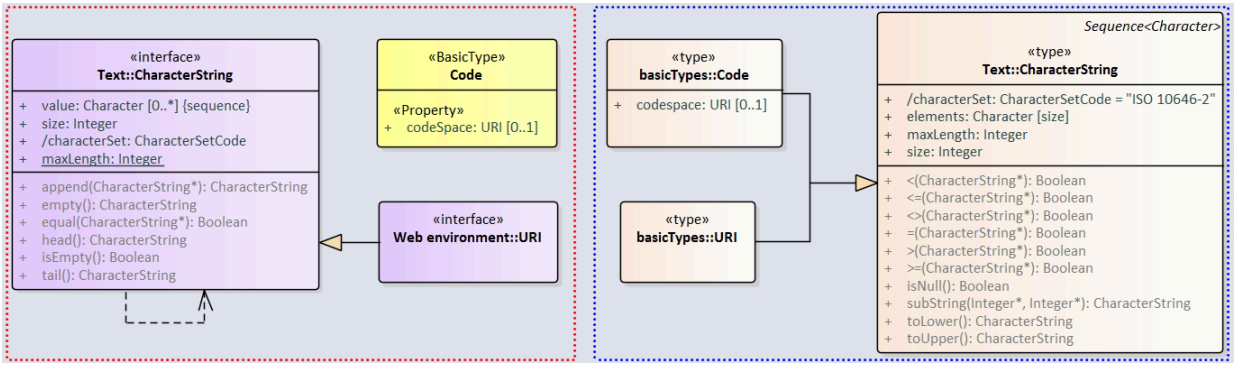


Figure 46 — CityGML 3.0 Part 1 — «BasicType» Code

The right-hand side submodel (inside of the blue dotted line) is also in the EAP, but part of the conceptual schema for ISO 19136:2007. While it would have been appropriate to reuse «type» basicTypes::Code from the ISO 19136:2007 conceptual schema rather than creating something novel, the CityGML 3.0 Conceptual Model could be revised to model class Code as an explicit subclass of «interface» CharacterString, resulting in «interface» Code.

In order to achieve maximum reuse and interoperability, in the present analysis the CityGML 3.0 Conceptual Model «BasicType» Code class was removed from the EAP and all references to it replaced by references to «type» basicTypes::Code.

OGC 20-010, Clause 9.2.3 defines «BasicType» MeasureOrNilReasonList as "a basic type that represents a list of double values and/or nil reasons together with a unit of measurement." In Figure 47 the upper submodel (inside of the red dotted line) is that in the EAP. The lower submodel (inside of the blue dotted line) is also in the EAP, but part of the conceptual schema for ISO 19136:2007. The parallel structure and identical class and property naming is evident. Also evident is the lack of identification in the CityGML 3.0 Conceptual Model for any allowed values for «CodeList» NilReasonEnumeration. The conceptual schema for ISO 19136:2007 is

clear that the concept is that of an extensible enumeration, not that of an empty code list to be later populated.

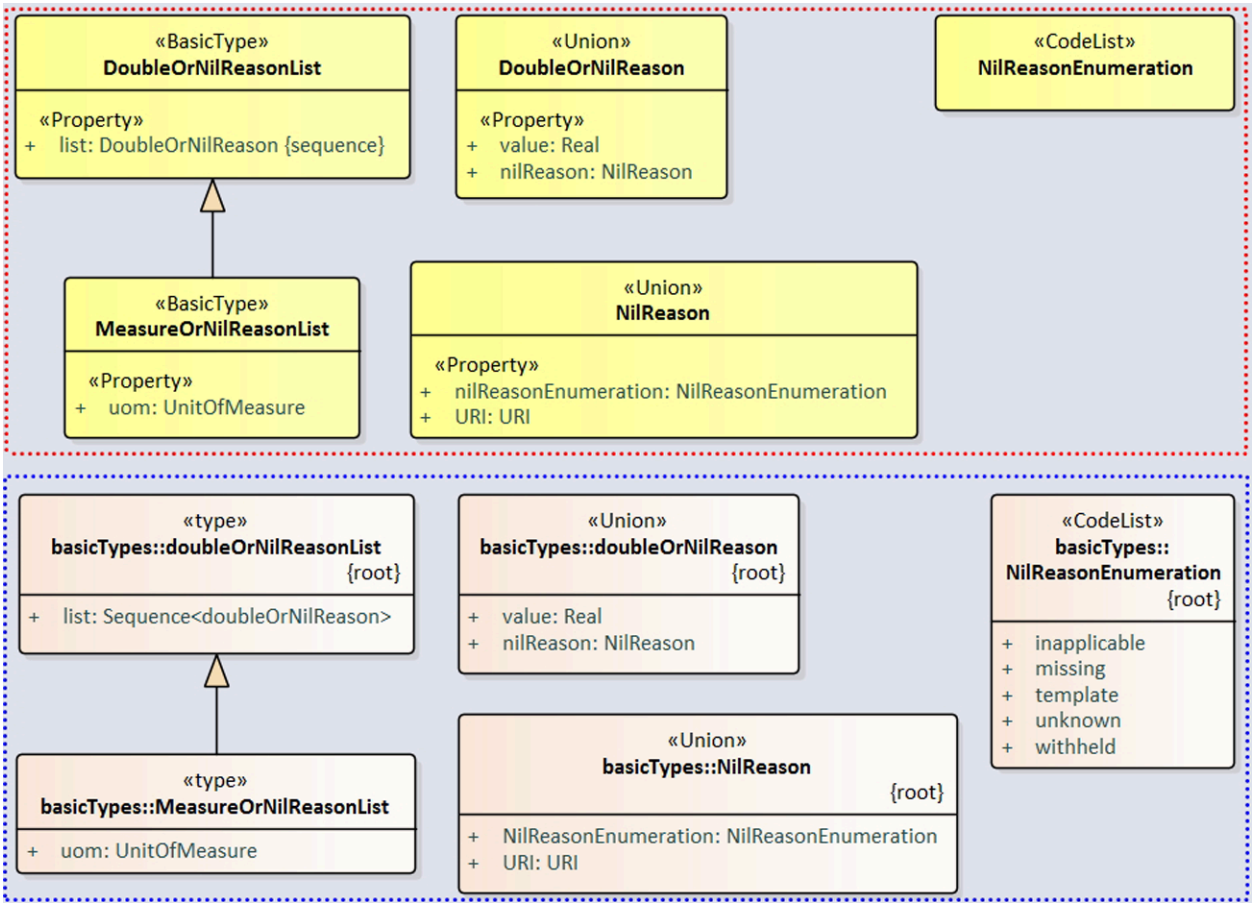


Figure 47 — CityGML 3.0 Part 1 — `«BasicType» MeasureOrNilReasonList`

While it would have been appropriate to reuse all five classes from ISO 19136:2007 conceptual schema rather than creating a novel replacement, the CityGML 3.0 Conceptual Schema could be revised to model `«CodeList»` NilReasonEnumeration` consistent with the existing class in ISO 19136:2007 by adding the five missing attributes (listed values).

In order to achieve maximum reuse and interoperability, in the present analysis all five CityGML 3.0 Conceptual Model classes were removed from the EAP and all references to them replaced by references to the corresponding classes in the “basicTypes” package in the ISO 19136:2007 conceptual schema.

RECOMMENDATION 10
/rec/mbs/reuse
When designing new conceptual models, maximize reuse of classes and submodels from existing standardized conceptual models/schemas in lieu of attempting to replicate the same capability de novo. Replication impedes interoperability and may inadvertently introduce subtle discrepancies or overt errors.

8.2.9. Lists of “basic type”

OGC 20-010, Clause 9.2.3, defines «BasicType» DoubleList as “an ordered sequence of double values.” It also defines «BasicType» DoubleBetween0and1List as “a basic type that represents a list of double values greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.”

Figure 48, from the EAP, documents both of these classes and their subclasses. Also documented (inside of the blue dotted line), and also in the EAP but part of the conceptual schema for ISO 19136:2007 , is «type» basicTypes::doubleList. It is evident that «BasicType» DoubleList is intended to avoid reuse of an existing type from the ISO 19136:2007 conceptual schema.

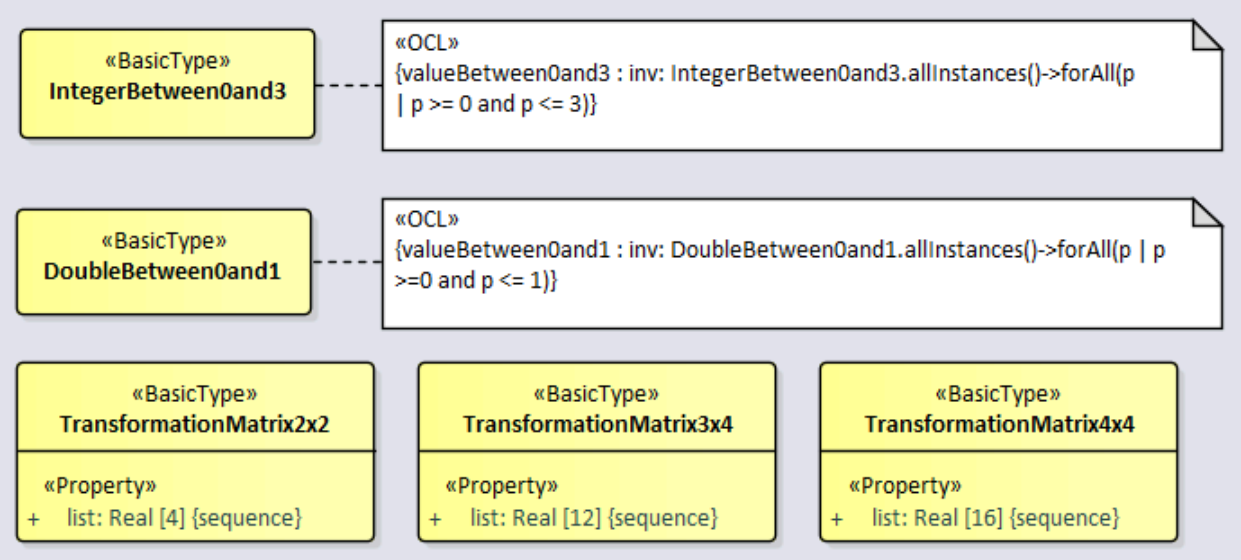


Figure 48 — CityGML 3.0 Part 1 — Core::DoubleList

In order to automatically derive a corresponding XML Schema specification using ShapeChange, the CityGML SWG made source code modifications to a former ShapeChange baseline. These modifications cannot be merged into the current ShapeChange baseline in their present state, and thus are not available to developers of Application Domain Extensions (ADE) based on the CityGML 3.0 Conceptual Model.

Because of the general utility of modeling lists of “basic types” in a consistent manner in conceptual models/schemas, the current ShapeChange software baseline (v2.10) was extended with one new XML Schema-related rule: rule-xsd-cls-basictype-list. This rule is independent of the CityGML 3.0 Conceptual Model, addressing the more general topic of the modeling of lists of “basic types” in Application Schemas using XML Schema. (Also see related rule: rule-xsd-cls-basictype)

The semantics of this new rule are that *a basic type that has a single property, where that property has a maximum multiplicity greater than 1* will be encoded as a list-based XML Schema simple type. The list item type is the XSD type of the UML property value type. If the minimum multiplicity of the UML property is 0 and the maximum multiplicity is unbounded (*), then the

length of the resulting list is not restricted. Otherwise, length restrictions are defined according to the multiplicity of the property.

The benefits of this rule are that a more natural approach to modeling lists in UML can be adopted, one that leverages the standard mechanisms for specifying UML properties (e.g., multiplicity) and thus avoids uses of Object Constraint Language (OCL) constraints in the Conceptual Model simply to add restrictions on the allowed list length. It also limits the necessity of employing validation mechanisms such as Schematron (for validating XML Instance documents) based on such OCL constraints in Conceptual Models.

The revised Concept Model no longer requires the inclusion of the «BasicType» DoubleBetween0and1List and «BasicType» DoubleList generalization classes; the revised five subclasses are easier to read and understand. The design pattern is simple and clear should developers of CityGML ADE desire to create new list-of-basic-type classes of their own. The revised CityGML 3.0 Conceptual Model appears as in Figure 49, noting that the special tagged-values added by the CityGML SWG in coordination with their revised ShapeChange software are no longer desirable and have been removed from all five list (formerly sub)classes in the Conceptual Model:

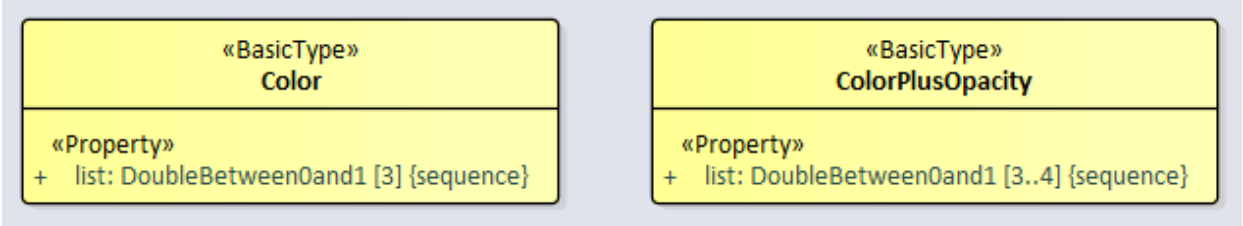


Figure 49-1 — Colors

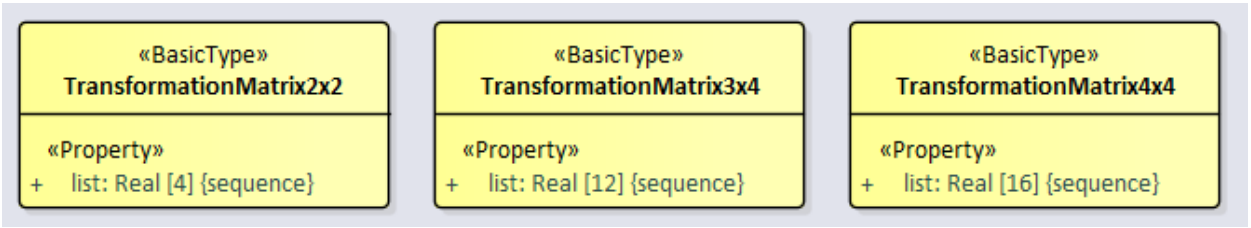


Figure 49-2 — Transformation matrices

Figure 49 — CityGML 3.0 Part 1 — Revised Model of Lists

When the new XML Schema-related rule-xsd-cls-basictype-list is applied (using ShapeChange) to this revised Conceptual Model, the following changes result in the generated XSD files:

1. Removed from cityGMLBase.xsd:

```
<simpleType name="DoubleBetween0and1ListType">
  <list itemType="core:DoubleBetween0and1Type"/>
</simpleType>
```

2. Revised in cityGMLBase.xsd:

```
<simpleType name="TransformationMatrix2x2Type">
  <restriction base="gml:doubleList">
```

```
    <length value="4"/>
  </restriction>
</simpleType>

<simpleType name="TransformationMatrix3x4Type">
  <restriction base="gml:doubleList">
    <length value="12"/>
  </restriction>
</simpleType>

<simpleType name="TransformationMatrix4x4Type">
  <restriction base="gml:doubleList">
    <length value="16"/>
  </restriction>
</simpleType>
```

Now as, respectively:

```
<simpleType name="TransformationMatrix2x2Type">
  <restriction>
    <simpleType>
      <list itemType="double"/>
    </simpleType>
    <length value="4"/>
  </restriction>
</simpleType>

<simpleType name="TransformationMatrix3x4Type">
  <restriction>
    <simpleType>
    <list itemType="double"/>
    </simpleType>
    <length value="12"/>
  </restriction>
</simpleType>

<simpleType name="TransformationMatrix4x4Type">
  <restriction>
    <simpleType>
      <list itemType="double"/>
    </simpleType>
    <length value="16"/>
  </restriction>
</simpleType>
```

3. Revised in `appearance.xsd`:

```
<simpleType name="ColorType">
  <restriction base="core:DoubleBetween0and1ListType">
    <length value="3"/>
  </restriction>
</simpleType>

<simpleType name="ColorPlusOpacityType">
  <restriction base="core:DoubleBetween0and1ListType">
    <minLength value="3"/>
    <maxLength value="4"/>
  </restriction>
</simpleType>
```

Now as, respectively:

```
<simpleType name="ColorType">
```

```
    <restriction>
      <simpleType>
        <list itemType="core:DoubleBetween0and1Type"/>
      </simpleType>
      <length value="3"/>
    </restriction>
  </simpleType>

  <simpleType name="ColorPlusOpacityType">
    <restriction>
      <simpleType>
        <list itemType="core:DoubleBetween0and1Type"/>
      </simpleType>
      <minLength value="3"/>
      <maxLength value="4"/>
    </restriction>
  </simpleType>
```

The generated XSD files, using the added rule and the current ShapeChange software baseline, are otherwise identical to those prepared by the CityGML SWG.

RECOMMENDATION 11
/rec/mbs/uml/basic-types
In general, adopt this clearer approach to modeling lists of “basic types” in conceptual schemas, and in particular revise the CityGML 3.0 Conceptual Model accordingly. Note that should the proposed change be made then CityGML ADE developers will be able to use the (now maintained) ShapeChange capability to generate their community-specific XSD encodings consistent with the CityGML 3.0 Conceptual Model.

8.2.10. Incompletely specified “basic types”

OGC 20-010, Table 11 defines three classes stereotyped «`BasicType`» that have not yet been addressed in this analysis; all are either incompletely or erroneously specified.

These classes are defined as follows:

- `ID` “is a basic type that represents a unique identifier.”
- `IntegerBetween0and3` “is a basic type for integer values, which are greater or equal than 0 and less or equal than 3. The type is used for encoding the LOD number.”
- `DoubleBetween0and1` “is a basic type for values, which are greater or equal than 0 and less or equal than 1. The type is used for color encoding, for example.”

They are documented in Figure 50. None of these three types is explicitly related to a type specified in ISO 19103:2015.

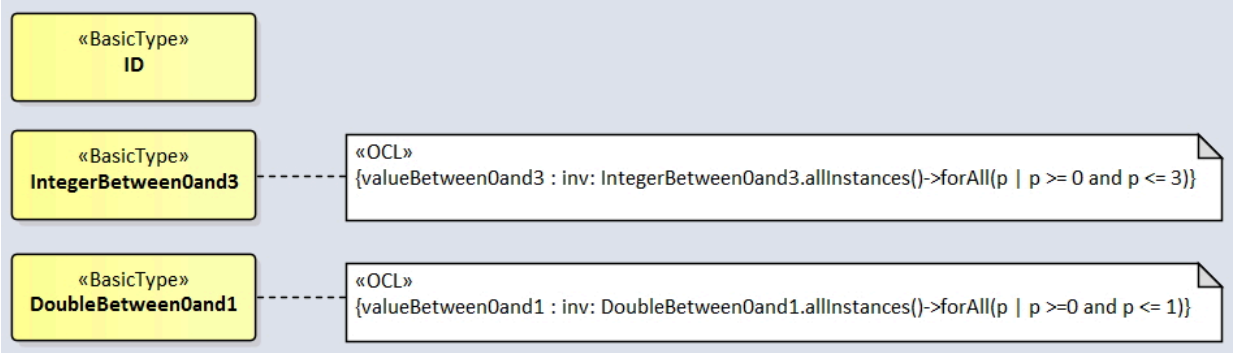


Figure 50 — CityGML 3.0 Part 1 — Remaining three "basic type" classes

It is uncertain whether an “ID” should be based on a Real, a CharacterString, or perhaps a URI. In the provisional XML Schema encoding for CityGML 3.0 it is implemented as `gml:id`, an XML Attribute whose type is `xs:ID`, a restriction of `xs:NCName` that is unique within the scope of a document. In the present analysis the CityGML 3.0 Conceptual Model `«BasicType» ID` class was assumed to be derived from `«type» CharacterString` in the ISO 19103:2015 conceptual schema.

The definition for “DoubleBetween0and1” doesn’t state the type of value that falls within the specified range (although the choice of class name does imply that, e.g., Decimal is not intended). In the present analysis the CityGML 3.0 Conceptual Model `«BasicType» DoubleBetween0and1` class was assumed to be derived from `«type» Real` in the ISO 19103:2015 conceptual schema.

The OCL expressions intended to constrain the range of allowed values of `IntegerBetween0and3` and `DoubleBetween0and1` are not valid. ISO/IEC 19507:2012 (OCL) states:

> The operation
>
> ```
> allInstances()
>    context Classifier
>    def: allInstances() : Set( T ) = -- all instances of self
> ```
>
> returns all instances of the classifier and the classifiers specializing it. May only be used for classifiers that have a finite number of instances. This is the case, for example, for user defined classes because instances need to be created explicitly, and for enumerations, the standard Boolean type, and other special types such as OclVoid. This is not the case, for example, for data types such as collection types or the standard String, UnlimitedNatural, Integer, and Real types.

A different approach to expressing the limited range of values of these two types is required; use of the UML tagged values rangeMinimum and rangeMaximum are a natural choice. Their assigned values can be easily exposed as an additional compartment in the presentations of these two classes in the EA class diagram.

Figure 51 documents the revised model for these three classes. Each class has two applied stereotypes; `«type»` is consistent with that of the superclass, `«BasicType»` serves as an

additional semantics-carrying indication specific to the CityGML 3.0 Conceptual Model (“a basic data type”).

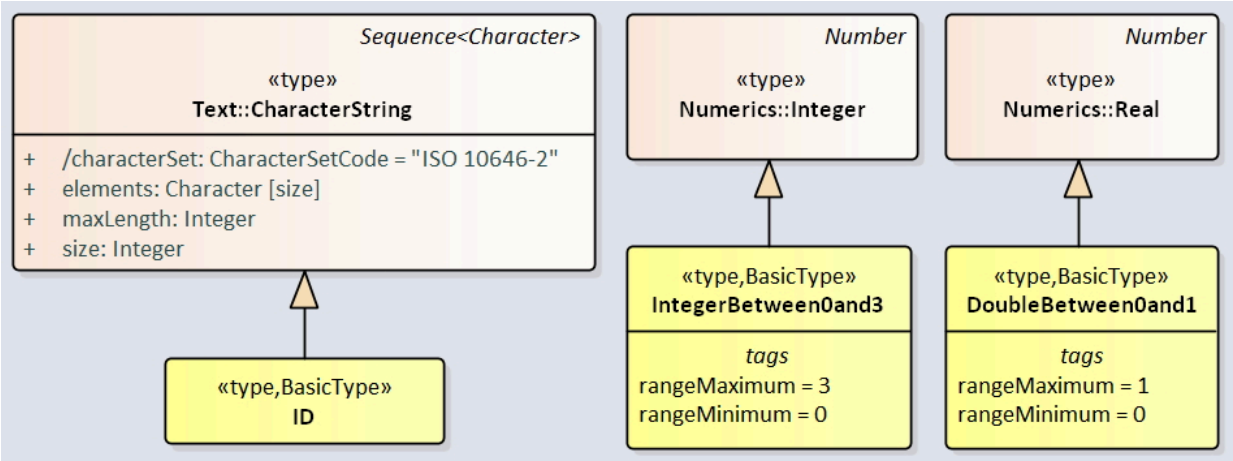


Figure 51 — CityGML 3.0 Part 1 — Revised "basic type" classes

RECOMMENDATION 12
/rec/mbs/uml/19103
Always construct (specialize, extend, compose) new types starting from those specified in ISO 19103:2015. Doing so enables maximum interoperability with, and reuse of, other conceptual models/schemas and their classes and/or components. It ensures that unambiguous platform-independent semantics are captured thus enabling support for a variety of technology-specific Implementation Specifications.

8.2.11. Revised Core Types Conceptual Model

Figure 52 summarizes the complete set of changes made to the core types in the CityGML 3.0 Conceptual Model. The three classes on the upper-left are from the ISO/TS 19103:2005 conceptual schema. The seven classes along the right edge are from the ISO 19136:2007 conceptual schema. The remaining eight classes are part of the (revised) CityGML 3.0 Conceptual Model.

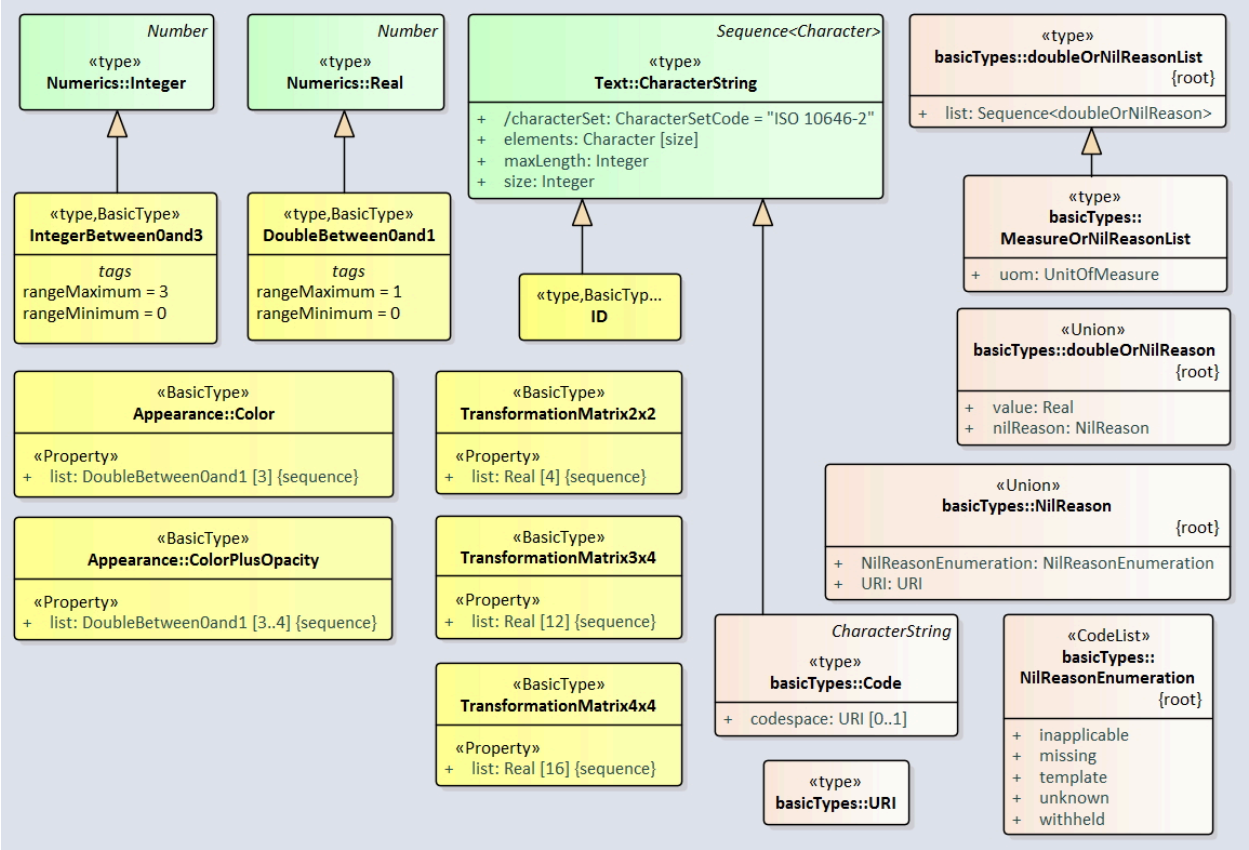


Figure 52 — CityGML 3.0 Part 1 — Final revised model

8.2.12. Constraints

OGC 20-010 documents fifteen (15) constraints. Of these, twelve are OCL invariant constraints that apply to classes within CityGML packages; the remaining three are Text invariant constraints that apply to class GM_Object (ISO 19107:2019).

Of the twelve OCL invariant constraints specified in the CityGML 3.0 Conceptual Model (EAP), seven have been replaced by alternative representations (using property multiplicity and tagged values), leaving five OCL invariant constraints. One of these constraints applies to an abstract class with four subclasses.

The constraint named `dataTypeOfValue` that is applied to class `Dynamizer::GenericTimeSeries` was determined to be different than the class-attached "note" appearing in the class diagram claiming to contain the content of that constraint. Annex F analyzes this inconsistency, determines the root cause, and illustrates several solutions.

The constraint named `dataTypeOfValue` was corrected, after which all constraints validated and eight corresponding Schematron assertions were successfully generated (see: https://shapechange.net/app-schemas/constraints/ and https://shapechange.net/targets/xsd/extensions/ocl/).

RECOMMENDATION 13
/rec/mbs/uml/ocl
Adopt a consistent approach to capturing and presenting constraints in class diagrams that ensures that for any given constraint it is both correctly characterized and that all of its presentations are synchronized. Lack of synchronization allows unrecognized errors to be present. Lack of proper constraint characterization limits the ability to check constraints for valid syntactic structure and logical use of (only) elements of the conceptual model. Lack of constraint validity limits (or precludes) their MDA-facilitated application in derived Implementation Standards. In addition to the three Text invariant constraints that apply to class GM_Object (ISO 19107:2003), external ISO 19100-series conceptual schemas upon which the CityGML 3.0 Conceptual Model asserts dependencies contained a further 40 informally specified Text invariant constraints. None of these informally specified constraints were evaluated.

8.2.13. Formal UML Profile

OGC 20-010 does not define a formal UML profile, although it creates and applies novel class and property stereotypes. Examination of the EAP reveals numerous tagged values being applied to packages, classes, and properties — although the significant majority are either unpopulated or contain default values specified by ISO 19136-1:2020 (GML).

Table 5 documents the assignments of tagged values to stereotypes, and for Boolean tagged values their default.

Table 5 — CityGML 3.0 Conceptual Model Informal UML Profile

STEREOTYPE	TAGGED VALUE	RANGE	DEFAULT
ApplicationSchema, Leaf	xsdDocument	String	<none>
ApplicationSchema	targetNamespace	String	<none>
ApplicationSchema	xmlns	String	<none>
ApplicationSchema	version	String	<none>
ApplicationSchema	gmlProfileSchema	String	<none>
ApplicationSchema	language	String	<none>
Leaf	xsdEncodingRule	notEncoded,iso19136_2007,gml33,citygml	<none>
TopLevelFeatureType, FeatureType, ObjectType	noPropertyType	Boolean	false

STEREOTYPE	TAGGED VALUE	RANGE	DEFAULT
TopLevelFeatureType, FeatureType, ObjectType	byValuePropertyType	Boolean	false
TopLevelFeatureType, FeatureType, ObjectType	isCollection	Boolean	false
TopLevelFeatureType, FeatureType	gmlMixin	Boolean	false
Union, DataType	noPropertyType	Boolean	false
Union	gmlAsGroup	Boolean	false
Union, DataType	xsdEncodingRule	notEncoded,iso19136_2007,gml33,citygml	<none>
BasicType	base	String	<none>
BasicType	length	Integer	<none>
BasicType	minLength	Integer	<none>
BasicType	maxLength	Integer	<none>
BasicType	rangeMinimum	String	<none>
BasicType	rangeMaximum	String	<none>
BasicType	xsdEncodingRule	notEncoded, iso19136_2007, gml33, citygml	<none>
Enumeration	<none>	CodeList	asDictionary
Boolean	true	CodeList	xsdEncodingRule
notEncoded, iso19136_2007, gml33, citygml	<none>	Property, Version	sequenceNumber
Integer	<none>	Property, Version	inlineOrByReference
inline, byReference, inlineOrByReference	inlineOrByReference	Property, Version	isMetadata
Boolean	false	Property	xsdEncodingRule

Notably absent are the tagged value `documentation` (with any stereotype of package, class, or property) as defined by the ISO 19136:2007 profile of UML, and the tagged values `definition` and `description` with the «`ApplicationSchema`» package and «`FeatureType`» class as defined by the ISO 19109:2015 profile of UML. Lack of support for, and employment of, tagged values

`definition` and `description` is especially pertinent to the scope of the CityGML 3.0 Part 1 Standard.

RECOMMENDATION 14
/rec/mbs/uml/best-practices-profile
Adopt a standards-development Best Practice UML Profile consisting of a coherent set of well-defined platform-independent stereotypes and associated tag definitions based on those currently specified by OMG UML 2.5, ISO 19103:2015, ISO 19109:2015, and ISO 19136: 2007. Deprecate the use of “notes” wherever possible in favor of the use of the “definition” and “description” tagged values. Encourage the use of formal OCL constraints rather than text-based constraints. The profile should emphasize both the specification of clear semantics in support of conceptual models and practical support for the specification of Implementation Standards (including use with MDA technologies).

8.3. DGGS conceptual model

8.3.1. Presentation style of UML packages

The DGGS MDS document uses a pre-defined template for representing UML packages.

Specifically, by using “automated inclusion”, for every UML package the Metanorma-LutaML plugin generates the following sections:

1. “{package name} Overview”
2. “Defining tables”

In “Defining tables”, all UML models that belong to this package will be individually enumerated, displaying the following attributes:

- Name
- Definition
- Stereotype
- Abstract
- Associations
- Public attributes

- Constraints (OCL)

A sample of the defining table is shown in Figure 53.

8.3.2. Defining tables

Table 53 — Elements of “Core Quantization Functions::DGG_Observation” (interface)

NAME:	DGG_Observation					
DEFINITION:	DGG_Observation is an abstract class holding ZonalIdentifier, OM_Observation tuples. In the context of Quantization, DGG_Observation holds records of Observations made with Data AssignmentProcess in assigning values to cells.					
STEREOTYPE:	interface					
ABSTRACT:	True					
ASSOCIATIONS:	*Association with*			*Obligation*	*Maximum occurrence*	*Provides*
	Zone				*	comprises
	OM_Observation			C	*	data
	Cell				*	comprises
PUBLIC ATTRIBUTES:	*Name*	*Definition*	*Derived*	*Obligation*	*Maximum occurrence*	*Data type*
	field	values that were observed		M	*	OM_Observation
	identifier	cells that were observed		M	*	ZonalIdentifier
CONSTRAINTS:	(none)					

Figure 53 — Sample "Defining table" from 20-040r3, 8.3.2

8.3.2. Annotation format in model editing tool

The DGGS conceptual model contains inconsistent annotation formats generally as described in Clause 8.2.1.

The DGGS annotation issue differs from that of CityGML in the following ways:

- The DGGS model utilized embedded AsciiDoc syntax within HTML and RTF, with some instances where HTML or RTF formatting is added to the AsciiDoc syntax, causing confusion on whether the encoded syntax is actually AsciiDoc.
- Some annotations in the DGGS model contained invalid AsciiDoc, leading to content incorporation failure when used in an MDS environment.

The recommendation here is to adhere to guidelines on how to encode a specific format within a model-editing tool's annotation editor, and apply syntax validation to that content in an external tool.

RECOMMENDATION 15
/rec/mbs/annotation-validation
Annotations present in information models are represented using a particular markup language. All annotations in the information models must be syntactically valid in order to facilitate the authoring of model-driven standards. If the chosen model authoring tool does not support validation of annotation syntax, an external validator must be used to assert the validity of annotation syntax.

8.3.3. Artifacts to be excluded

As noted in Clause 4.13, it proved necessary to exclude objects from the PIM in the MDS.

In the DGGS conceptual model located within the ISO/TC 211 HMM, there exists a convention to exclude certain objects from the MDS but are present inside the EAP model.

The explicit exclusion is achieved via the following naming conventions:

- UML packages that have a name that begins with the prefix `old:`
- Diagrams (figures) that have a name prefix of `fyi:` or `old:`
- All objects under the UML package with name `Spare`

In the prototype, a filtering function is implemented to skip objects that match the criteria above.

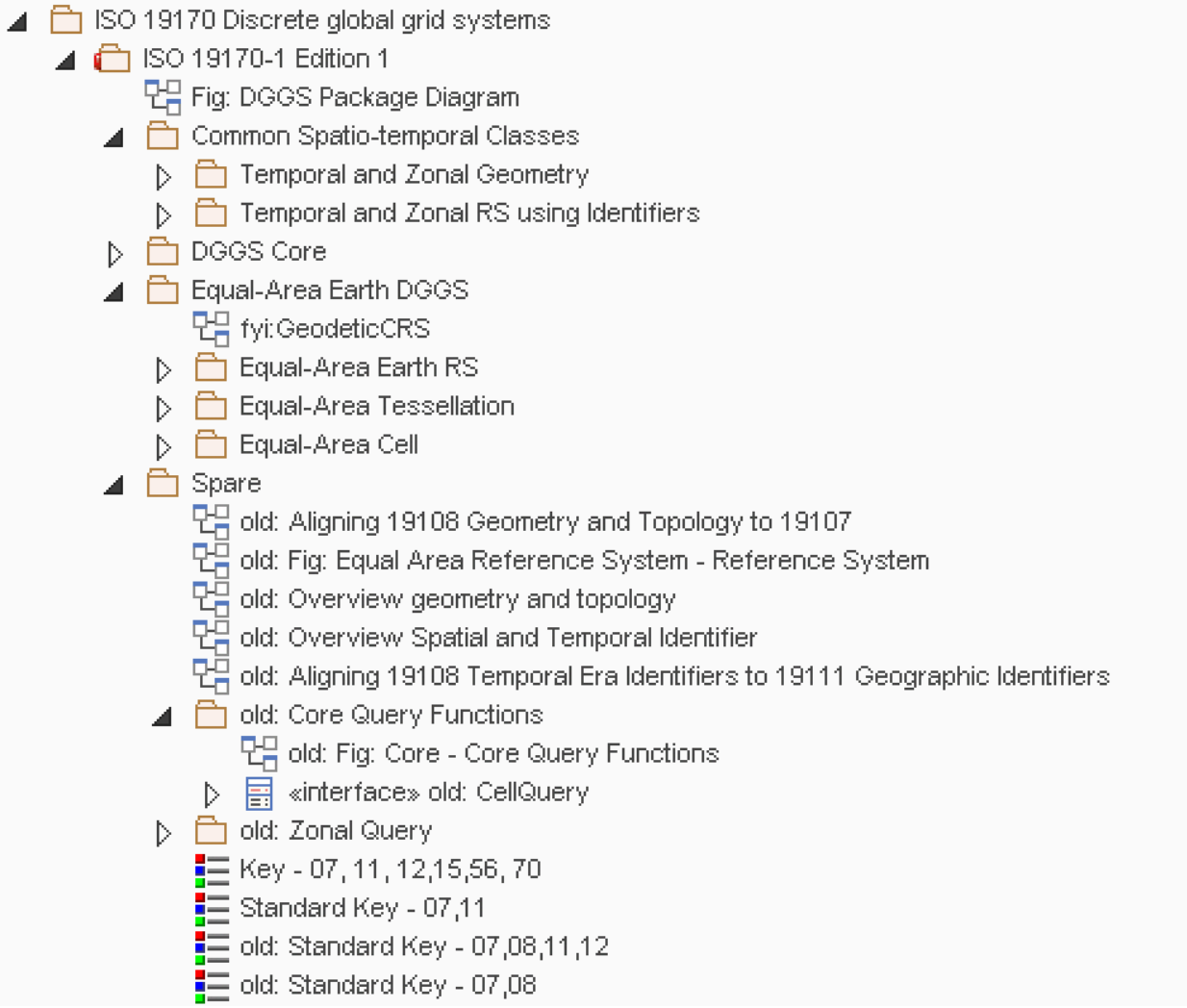


Figure 54 — HMMG DGGS package — Excluded objects prefixed with `old:` `fyi:` and the `Spare` package

This exclusion mechanism is crucial for UML packages that intends to have the MDS clause structure fully and automatically mirror the UML package hierarchy.

In a document like CityGML where the MDS clause hierarchy is manually derived (through explicit inclusion, instead of automatic inclusion), exclusion specification is unnecessary.

In practice, exclusion can be achieved using one or both ways:

1. Specified in the information model, similar to the naming convention exclusion implemented in the DGGS conceptual model,

2. Specified in the model-driven standard, through filtering mechanisms implemented in the CityGML conceptual model.

The choice of the exclusion mechanism depends on several factors:

- model authoring tools do not often provide version management features, causing draft models to be stored in the same manner as published models. In this case the draft models (or inactive models) should be marked with a naming convention that indicates their inactive status.
- Models used for MDS should typically only contain normative content. Any informative content should be clearly marked within the information model using a convention.

RECOMMENDATION 16
/rec/mbs/exclusion
Information models can contain content that is not intended to be incorporated into a model-driven standard. If a model-driven standard automatically detects and incorporates content from an information model, and that the information model contains content to be excluded, the items to be excluded must be explicitly defined. When a model-driven standard uses a UML package that contains content to be excluded, such as UML models, diagrams and packages, the exclusion of objects should be defined with a naming convention.

8.3.4. References to MDS content generated from model

As described in Clause 4.17.3, references from within the model to document artifacts, such as figures, pose a particular challenge for model management, as they introduce a dependency on external tools.

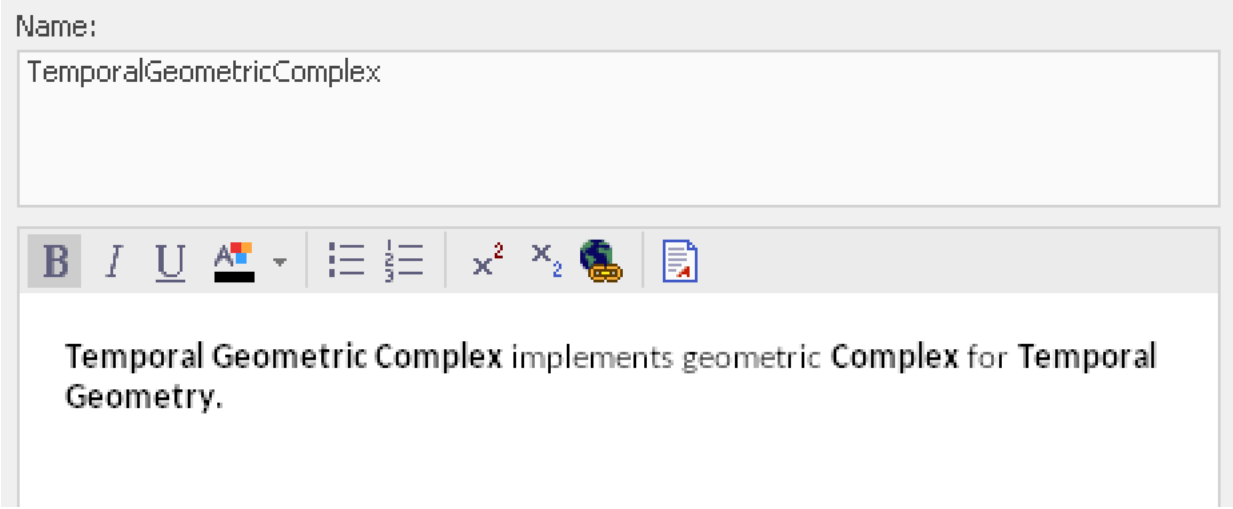


Figure 55 — HMMG DGGS package — Example of annotations containing references to other model objects

8.3.5. References to MDS content elements (figures, tables, clauses)

As described in Clause 4.17.3, references from within the model to document artifacts, such as figures, pose a particular challenge for model management, as they introduce a dependency on external tools.

One option is to ignore the dependency, and keep the cross-references as static text. The problem this introduces is that figures and other assets in the target document can and will be renumbered, as content is added to and rearranged. This situation has been seen in the DGGS conceptual model.

If the cross-references from the model annotations are kept as static text, they will need to be updated manually every time the target document changes; and because the model authoring tools are not bidirectionally integrated with the authoring environment, any such updates will be particularly laborious.

NOTE: The "Replace All" cannot be used, and even if it could, it will not search both the target document and the information model annotations in a single pass.

For any but a trivial number of cross-references from the information model, the cross-references need to be dynamic, and able to exploit auto-numbering.

RECOMMENDATION 17
/rec/mbs/autonum
Where feasible, cross-references in annotations authored within model authoring tools such as Enterprise Architect need to be marked up in the same markup language as that used in MDA (i.e. Metanorma AsciiDoc, as already recommended in Clause 8.2.1, Recommendation 1: `/rec/mbs/annotation-format`). Those cross-references need to be dynamic, pointing to anchors in the target document, so that they can be updated as their referents are auto-numbered. A consistent referencing protocol will need to be adopted, to cope with the fact that models can drive more than one target MDS document, and that references may be resolved either within the current document, or an external document. Metanorma has already needed to address this concern in processing document collections, involving documents cross-referencing each other, where it is not necessarily known at authoring time whether the reference target is in the same document or another document: https://www.metanorma.org/author/topics/document-format/collections/#indirect-xrefs

The annotations of a model may include content such as tables and figures which would be subject to autonumbering in a normal document. If that is the case, those assets need to be so marked up that they will be autonumbered in their proper order — in the case of Metanorma, when the Metanorma document is assembled out of the derived and supplementary truth mediated via LutaML. This is a consistent outcome of Clause 8.2.1, Recommendation 1: `/rec/mbs/annotation-format`.

If this is not done, then content contributed as derived truth (e.g. figures in model annotations) is not subjected to the same formatting rules as content contributed as supplementary truth (e.g.

figures added within the target document), and the document becomes inconsistent and poorly formatted.

8.3.6. Utilize tagged values

The DGGS conceptual model contains some idiosyncratic syntax in UML model attributes that should be normalized.

For example, under “Elements of Core Topological Query Functions::ZoneQuery”, there is OCL used to indicate the “dimension” of an operation, such as `<<query>> (1D)`, used to indicate that the data type of a relative position should be one-dimensional.

However, this was conflicting with `<<query>>` (for a two-dimensional relative position), and was using a parenthetical qualifier to restrict the type of the UML attribute.

This instance was compromising the processing of the UML model, and needed to be resolved by differentiating the type token `1D-directPosition` from `directPosition`. A clean separation of concerns needs to be part of all identifiers used in well-managed information models.

UML does provide the capability for entering structured data in form of “tagged values”. It is strongly recommended that no structured data elements are entered as free-form or undifferentiated text within the model authoring tool.

8.4. Specification-friendly UML package names, name uniqueness

In an MDA approach, a common operation is to find a particular model using its namespaces. In the case of the transformation of DGGS PIM into an MDS, the specification of a fully-qualified name for UML models is a common sight.

However, within the DGGS conceptual model, most UML packages utilize long names that include spaces and punctuation characters, such as “Common Spatio-Temporal Classes”.

These names prioritize human-readability but present a challenge to machine-referencing.

The useability challenge of these long names is compounded when accompanied by a hierarchical UML package structure. When using UML models with MDA, the models need to be referred to via fully qualified names given the possibility of conflicting model names across UML packages. These fully qualified references are not only cumbersome for MDS and PSM authors, but are also uncomfortably long for any human audience.

Perhaps more importantly, it is beneficial to have unique UML model names within the top-level UML package, so that the chance of confusing two UML models from different UML packages would be minimal.

9

TESTBED SUB-TASK D143

9 TESTBED SUB-TASK D143

9.1. General

The D143 sub-task was run by SURROUND Australia Pty Ltd.

D143 presents a modelling approach and a series of investigations into issues encountered when trying to implement an OWL-based (W3C OWL) "Semantic Web" approach to MDS.

The approach was undertaken as an alternative approach to UML-driven MDS which is the current method being employed by the OGC and related organizations such as ISO/TC 211.

9.2. Introduction

9.2.1. Refocused Deliverable

Work for this Testbed 17 Deliverable 143 was originally anticipated to deliver a:

> software tool capable of exercising UML model-to-platform specific model transformation for the work items identified in the Transformation Exercises table

However, as per SURROUND Australia Pty Ltd's (SURROUND's) response to the Testbed, outlined in its Statement of Work (SURROUND SoW), an approach to the generation of Standards' documents "directed by the World Wide Web Consortium (W3C)'s *Profiles Vocabulary*" (PROF) (W3C TR dx-prof) and that is "provenance and profile dependencies"-aware was undertaken, rather than a tool build directly. This was for two reasons:

1. The OGC has an embedded tool-based approach (see Testbed 17, D135: https://www.ogc.org/projects/initiatives/t17)
2. A quick assessment by this author indicated that to model-drive a whole Standard, and not just the *Domain Model*[1], lots more Standard element modelling was required

The focus on PROF was undertaken because it provides a formal, Semantic Web (thus model-driven) vocabulary to "relate profiles (standards that profile other standards) to one another and

[1]See next section for a description of this term

to link profile parts to one another". This then provides "several options for describing PIM/PSM relations".

The form of deliverables for this Deliverable is now a Report — this document — and a series of technical resources — ontologies, code examples and images — that are listed in the next section.

9.2.2. Deliverable Resources

- Report — this section
- Additions to the remainder of this MDS Engineering Report
- oMDS tool code — uploaded to https://gitlab.ogc.org/ogc/T17-D143-MDA-Tool-1

9.2.3. Standard relations modelling & scenario selection

As per the SURROUND SoW, two potential options were presented for relating Platform-Independent standard elements and Platform-Specific ones (PIM/PSM):

1. Declaring the PIM and any PSMs to be parts of a single, conceptual, Standard
2. Defining the PIM and each PSM as a Standard and relating them with PROF properties

PROF provides a small vocabulary of roles that elements of a Standard might play W3C TR dx-prof, Clause 9 that could be used in scenario 1 to indicated PIM/PSM relations. Before this work, no assessment of whether these roles were related to PIM/PSM Standard concerns had been conducted. PROF provides only one Standard-to-Standard relationship, `isProfileOf`, that can be used to related multiple standards, as per scenario 2. There has been some talk of creating other relationships or specialized forms of `isProfileOf` but this has not happened yet.

For this Report, scenario 1 only is considered further for this reason: PIMs, PSMs and other Standards' elements seem to be easily considered parts of a conceptual standard, and more modelling skill, in PROF, is available to describe and relate them than to relate multiples Standards together. Also, things modelled as per scenario 1 may be able to be re-implemented according to scenario 2 at a later date if, for instance, someone wishes to take a part of a standard out of that standard and give it it's own life. So if scenario 1 can be made to work, scenario 2 could be implemented later.

9.2.4. Exemplar selection

The SoW indicated that the development of the OGC's GeoSPARQL standard' 1.1 version (OGC GeoSPARQL 1.1) would be used to exemplify the use of PROF. Within this Testbed activity, the Australian/New Zealand 3D Cadastre Standard Sample has also been used to demonstrate things.

These exemplars will be referred to throughout as *GeoSPARQL 1.1* and *3D Cad*.

9.2.5. Additional work

Annex H regarding the relationships between RDF graphs and Property Graphs is included in this document as this was generated in response to questions on that topic within the Testbed.

9.3. Standard modelling

The D143 deliverable produced a methodology and did not produce a "software tool" (as stated in Clause 9.2.1). The methodology starts with modelling the elements of a Standard.

9.3.1. PROF Standards' elements modelling

PROF provides model (ontology) elements to describe and relate parts of a Standard. This mostly involves defining basic properties of the elements for technical access (format etc.), but also the Role or Roles that the various parts play with PROF providing a roles vocabulary (W3C TR dx-prof, Clause 9) that contains:

- Constraints
- Example
- Guidance
- Mapping
- Schema
- Specification
- Validation
- Vocabulary

Definitions for these Roles are given by PROF, and PROF encourages uses of it to extend this vocabulary for their own purposes. SURROUND has extended this vocabulary with a couple of additional roles for this work and other projects. That extended vocabulary can be seen at:

- https://data.surroundaustralia.com/def/prof-roles

So far, the only additional Roles are:

- Repository

Since GeoSPARQL 1.1 has had its elements modelled according to PROF and had them assigned these roles, some of its elements and their roles are given as an example of this modelling in Figure 57 using PROF's diagramming notation, a key of which is given in Figure 56.

Figure 56 — Figure Element Key

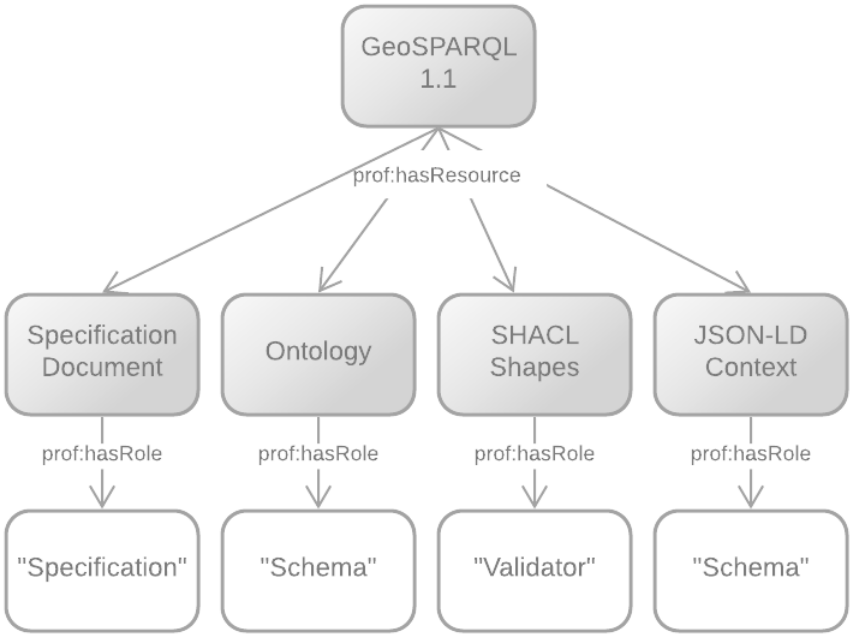


Figure 57 — GeoSPARQL 1.1 Profile elements, modelled according to PROF, using figure elements from Figure 56

Note that, according to PROF, the thing that most people see as "The Standard" is termed a *Specification* (document) and that there are other, potentially equally important, elements of the Standard, such as *Schema* — technical data model descriptions, *Validators* — technical assets to validate data claiming conformance to the Standard, and so on.

There is no direct handling of PIM or PSM elements in the PROF Roles vocabulary, however:

- such terminology could be introduced in a Roles vocabulary extension, such as SURROUND's linked to above
- we may infer that a "Specification" is somewhat abstract (PIM) and that "Schema" is not (PSM)

With this modelling in mind, the next section considers PIM/PSM modelling.

9.3.2. PROF-based PIM/PSM modelling

As stated above, PROF's Roles for Resources don't explicitly model PIM & PSM concerns within Standards and neither do other PROF elements (classes or properties), however, at least two forms of PIM/PSM splits were foreseen at the start of the Testbed 17 MDS project, as indicated in the section titled 'Refocused Deliverable' above:

1. Declaring the PIM and any PSMs to be parts of a single, conceptual, Standard
2. Defining the PIM and each PSM as a Standard and relating them with PROF properties

9.3.3. Approach 1: PIM & PSM parts in a single Standard

For Approach 1, the PIM and various PSMs of an imagined *Spatial Standard* could be declared using the following RDF conforming to PROF:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX prof: <http://www.w3.org/ns/dx/prof/>
PREFIX role: <http://www.w3.org/ns/dx/prof/role/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema=>
PREFIX : <http://example.com/def/spatialstandard/>
<http://example.com/def/spatialstandard>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard"@en ;
    dcterms:description "Profile declaration of the Spatial Standard"@en ;
    prof:hasResource
        :pim-spec ,
        :td-rdf ,
        :psm-json-schema ,
        :psm-json-ld ,
        :psm-xml ,
        :psm-xml-validator ;
.

:pim-spec
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard Specification"@en ;
    dcterms:description "This Standard's normative, Platform Independent
Model"@en ;
    dcterms:conformsTo <https://www.ogc.org/standards/modularspec> ;
    dcterms:format "application/pdf"^^dcterms:IMT ;
    prof:hasRole role:specification ;
    prof:hasArtifact "http://somewhere.com/resources/3"^^xsd:anyURI ;
.

:td-rdf
    a prof:ResourceDescriptor ;
    dcterms:title "Terms & Definitions Vocabulary"@en ;
    dcterms:description "This Standard's PIM's Terms & Definitions presented
as a SKOS Vocbaulary, in RDF (machine-redable)"@en ;
```

```
    dcterms:conformsTo <https://w3id.org/profile/vocpub> ;
    dcterms:format "text/turtle"^^dcterms:IMT ;
    prof:hasRole role:vocabulary ;
    prof:hasArtifact "http://somewhere.com/resources/12"^^xsd:anyURI ;
.

:psm-json-schema
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard JSON Schema"@en ;
    dcterms:conformsTo <https://datatracker.ietf.org/doc/draft-bhutton-json-
schema/00/> ;
    dcterms:format "application/schema+json"^^dcterms:IMT ;
    prof:hasRole role:schema ;
    prof:hasArtifact "http://somewhere.com/resources/101"^^xsd:anyURI ;
.

:psm-json-ld
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard JSON-LD Schema"@en ;
    dcterms:conformsTo <https://www.w3.org/TR/json-ld11/> ;
    dcterms:format "application/ld+json"^^dcterms:IMT ;
    prof:hasRole role:schema ;
    prof:hasArtifact "http://somewhere.com/resources/36"^^xsd:anyURI ;
.

:psm-xml
    a prof:ResourceDescriptor ;
    dcterms:title "Spatial Standard XML Schema"@en ;
    dcterms:format "application/xm"^^dcterms:IMT ;
    prof:hasRole role:schema ;
    prof:hasArtifact "http://somewhere.com/resources/27"^^xsd:anyURI ;
.

:psm-xml-validator
    a prof:ResourceDescriptor ;
    dcterms:title "XML Validator"@en ;
    dcterms:format "application/xm"^^dcterms:IMT ;
    prof:hasRole role:validator ;
    prof:hasArtifact "http://somewhere.com/resources/55"^^xsd:anyURI ;
.
```

Figure 58 — PROF RDF for PIM and PSMs of an imagined single *Spatial Standard*

Figure 59 below is an informal diagram of the elements of the RDF code above.

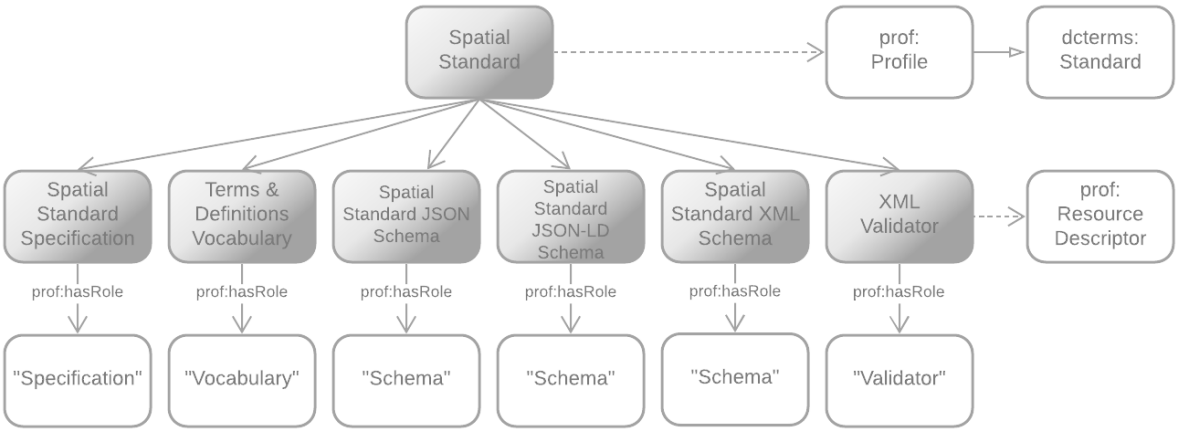


Figure 59 — Spatial Standard example, modelled according to PROF as parts (`Resource` instances) within a single `Standard`, using figure elements from Figure 56

In the dummy "Spatial Standard" code above, the Standard is identified as a conceptual thing of type (of OWL Class) `dcterms:Standard` and not as any particular resource, such as a Specification. A series of resources are linked to the conceptual Standard and described using PROF's `ResourceDescriptor` class which is made for this purpose. Using the PROF Roles listed in Clause 9.3.1, a `:pim-spec` object is given, with role `role:specification`, a `:psm-xml-validator` object with role `role:validator` etc. Here we may infer that the Specification resource is indeed a Platform Independent Model and we may expect it to be a document (its format is given as PDF) with at least a *Domain Model* and likely a number of other, common, elements, such as a *Terms & Definitions* section too. See Clause 9.3.6 for a description of Specification element modelling.

The *Terms & Definitions* content is also available in machine-readable form (RDF, according to the *VocPub*[2] profile of W3C TR skos-reference, and this is hard to classify as either a PIM or a PSM thing.

A JSON Schema (`:psm-json-schema`), a JSON-LD (`:psm-json-ld`) and an XML (`:psm-xml`) Schema are all clearly PSMs. The XML Schema schema is also accompanied by an XML validator (`:psm-xml-validator`).

With this way of modelling PIM & PSM objects using PROF roles, it is relatively easy to identify a PIM-like resource — the Specification — and PSM-like resources — various Schemas, but there are other elements too — the Vocabulary and the Validator — that PROF enables the identification and description of, and which are common to find in real-world Standards — but which do not easily lend themselves to PIM or PSM classification.

There are several options for formally identifying Standard elements described in this way using PROF as PIM or PSM things:

[2] https://w3id.org/profile/vocpub

1. Create PIM & PSM PROF Roles

    - resources described using PROF may have more than one Role
    - the `:psm-json-schema` described above might have both the roles of `role:schema` and `ex:psm`

2. Create a PROF extension for PIM/PSM descriptions

    - PROF could be extended with new properties to describe PIM/PSM attributes
    - `:psm-json-schema` above could then have a PROF Role of `role:schema` and an orthogonal property indicating PIM/PSM classification, perhaps `ex:mdsRole ex:psm`

9.3.4. Approach 2: PIM and each PSM as separate Standards

Information similar to that given in the section above could be declared for a PIM and a PSMs that are described as separate standards but related to one another. This may look as follows:

```
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX prof: <http://www.w3.org/ns/dx/prof/>
PREFIX role: <http://www.w3.org/ns/dx/prof/role/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema=>
PREFIX : <http://example.com/def/spatialstandard/>
<http://example.com/def/spatialstandard>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard"@en ;
    dcterms:description "Profile declaration of the Spatial Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard Specification"@en ;
        dcterms:description "This Standard's normative, Platform Independent
Model"@en ;
        dcterms:conformsTo <https://www.ogc.org/standards/modularspec> ;
        dcterms:format "application/pdf"^^dcterms:IMT ;
        prof:hasRole role:specification ;
        prof:hasArtifact "http://somewhere.com/resources/3"^^xsd:anyURI ;
    ] ,
    [
        a prof:ResourceDescriptor ;
        dcterms:title "Terms & Definitions Vocabulary"@en ;
        dcterms:description "This Standard's PIM's Terms & Definitions
presented as a SKOS Vocbaulary, in RDF (machine-redable)"@en ;
        dcterms:conformsTo <https://w3id.org/profile/vocpub> ;
        dcterms:format "text/turtle"^^dcterms:IMT ;
        prof:hasRole role:vocabulary ;
        prof:hasArtifact "http://somewhere.com/resources/12"^^xsd:anyURI ;
    ]
.

<http://example.com/def/spatialstandard-json>
    a prof:Profile , dcterms:Standard ;
```

```
    dcterms:title "Spatial Standard in JSON"@en ;
    dcterms:description "Profile declaration of the JSON PSM of the Spatial
Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard JSON Schema"@en ;
        dcterms:conformsTo <https://datatracker.ietf.org/doc/draft-bhutton-
json-schema/00/> ;
        dcterms:format "application/schema+json"^^dcterms:IMT ;
        prof:hasRole role:schema ;
        prof:hasArtifact "http://somewhere.com/resources/101"^^xsd:anyURI ;
    ] ;
    prof:isProfileOf <http://example.com/def/spatialstandard> ;
.

<http://example.com/def/spatialstandard-json-ld>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard in JSON"@en ;
    dcterms:description "Profile declaration of the JSON PSM of the Spatial
Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard JSON-LD Schema"@en ;
        dcterms:conformsTo <https://www.w3.org/TR/json-ld11/> ;
        dcterms:format "application/ld+json"^^dcterms:IMT ;
        prof:hasRole role:schema ;
        prof:hasArtifact "http://somewhere.com/resources/36"^^xsd:anyURI ;
    ] ;
    prof:isProfileOf <http://example.com/def/spatialstandard-json> ;
.


<http://example.com/def/spatialstandard-xml>
    a prof:Profile , dcterms:Standard ;
    dcterms:title "Spatial Standard in XML"@en ;
    dcterms:description "Profile declaration of the XML PSM of the Spatial
Standard"@en ;
    prof:hasResource [
        a prof:ResourceDescriptor ;
        dcterms:title "Spatial Standard XML Schema"@en ;
        dcterms:format "application/xm"^^dcterms:IMT ;
        prof:hasRole role:schema ;
        prof:hasArtifact "http://somewhere.com/resources/27"^^xsd:anyURI ;
    ] ,
    [
        a prof:ResourceDescriptor ;
        dcterms:title "XML Validator"@en ;
        dcterms:format "application/xm"^^dcterms:IMT ;
        prof:hasRole role:validator ;
        prof:hasArtifact "http://somewhere.com/resources/55"^^xsd:anyURI ;
    ] ;
    prof:isProfileOf <http://example.com/def/spatialstandard> ;
.
```

Figure 60 — PROF RDF for PIM and PSMs of *Spatial Standard*, modelled as separate related `Standard` instances

Figure 61 below is an informal diagram of the elements of the RDF code above.

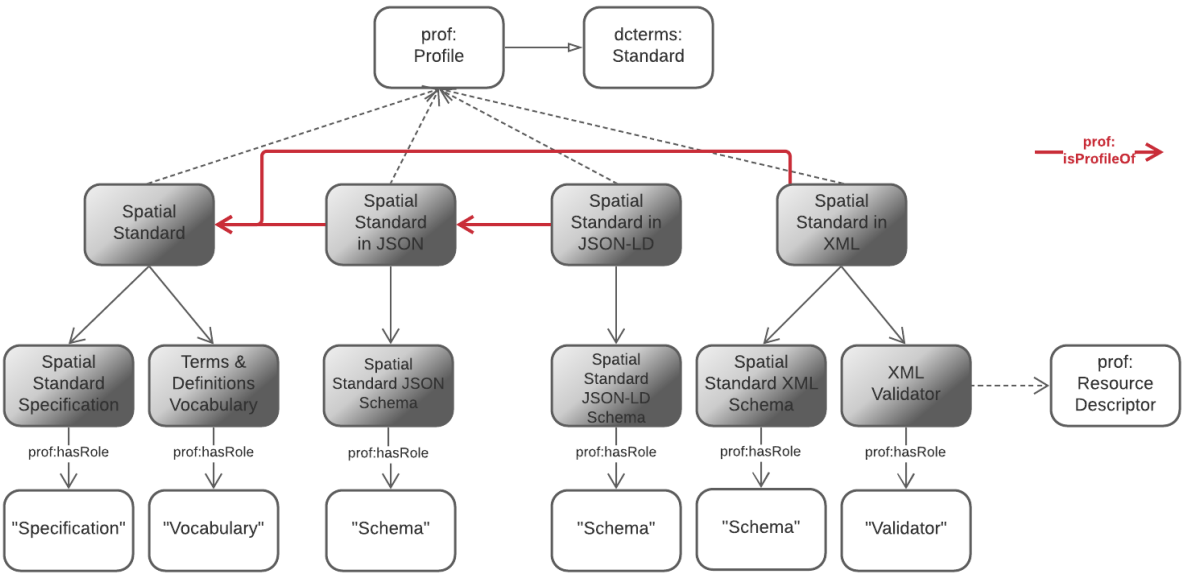


Figure 61 — Spatial Standard example, modelled according to PROF as multiple `Standard` instances with profile relations between them, using figure elements from Figure 56

Above, four Standards are declared — `http://example.com/def/spatialstandard`, `http://example.com/def/spatialstandard-json`, `http://example.com/def/spatialstandard-json-ld` & `http://example.com/def/spatialstandard-xml` - with the second and fourth being shown to be profiles of the first (the JSON & XML PSMs of the PIM) and the third to be a profile of the second (the JSON-LD PSM of the JSON PSM). Currently there is only one semantic way of relating multiple Profiles/Standards to one another in PROF and that is by using `prof:isProfileOf`, as is done in these cases. Note that we can learn, with this way of working, that there is a relationship between the JSON & JSON-LD PSM forms.

This method of characterization follows the ISO/TC 211 pattern of having a so-called Dash One ("-1") conceptual (PIM) standard with Dash two, Three etc ("-2", "-3" etc.) for various encodings.

For Standards modelled in this way, PIM/PSM identification could be achieved by:

1. Using method 2. from Clause 9.3.3
   - "Create a PROF extension for PIM/PSM descriptions"
   - applied to Standards, not resources described within them
2. Specialization of the `prof:isProfileOf` property
   - perhaps a property such as `ex:isPlatformSpecificProfileOf` could be used to link a PSM to a PIM

Method 1 from Clause 9.3.3 — "Create PIM & PSM PROF Roles" — could not be used in this modelling as the Roles apply to parts within a Standard, not across Standard instances. There

is no sensible way to define Roles in an Open World modelling of multiple Standard instances, since there is no defined envelope in which all the Standard instances exist, unlike resources within a Standard, thus there is no way to indicate what the role of a Standard is with respect to.

9.3.5. Approach review

Both the approaches above have merit. The first approach — Parts — is easier to implement but perhaps more constrained: all the PIM & PSM elements must conceptually be within a single Standard.

It may be that Approach 2 is required for real world implementation, given the fitful nature of standards development (not all the parts, PIMs & PSMs etc. are developed in a coordinated manner).

The complete examples of modelled Standards in this report in Clause 9.3.9 and Clause 9.3.10 use Approaches 1 and 2 respectively. This is due to requirements set outside of this Testbed activity which required GeoSPARQL to be one Standard and ANZ 3D Cad to deliver distinct *Conceptual* and *Implementation* Standards which are essentially equivalent to PIMs and PSMs.

9.3.6. Specification elements

To sensibly model Standards according to PROF, we need to be able to model the parts of a Standard's Specification.

The Specification document of a Standard is often considered to be THE Standard. PROF modelling deliberately separates the conceptual Standard from the concrete Specification.

A conjecture is posed which is that:

> Different parts of a Standard's Specification are best modelled with different model forms

For example, the conceptual area of the Standard's concern, which is termed here a *Domain Model*, is likely best modelled with a form of ontology, perhaps an OWL model or a UML Class model, however the *Terms & Definitions* section often found in Specifications is likely best modelled with a taxonomy model, perhaps W3C TR skos-reference.

Figure 62 indicates the parts in a number of Specifications and proposed forms of model to model their content.

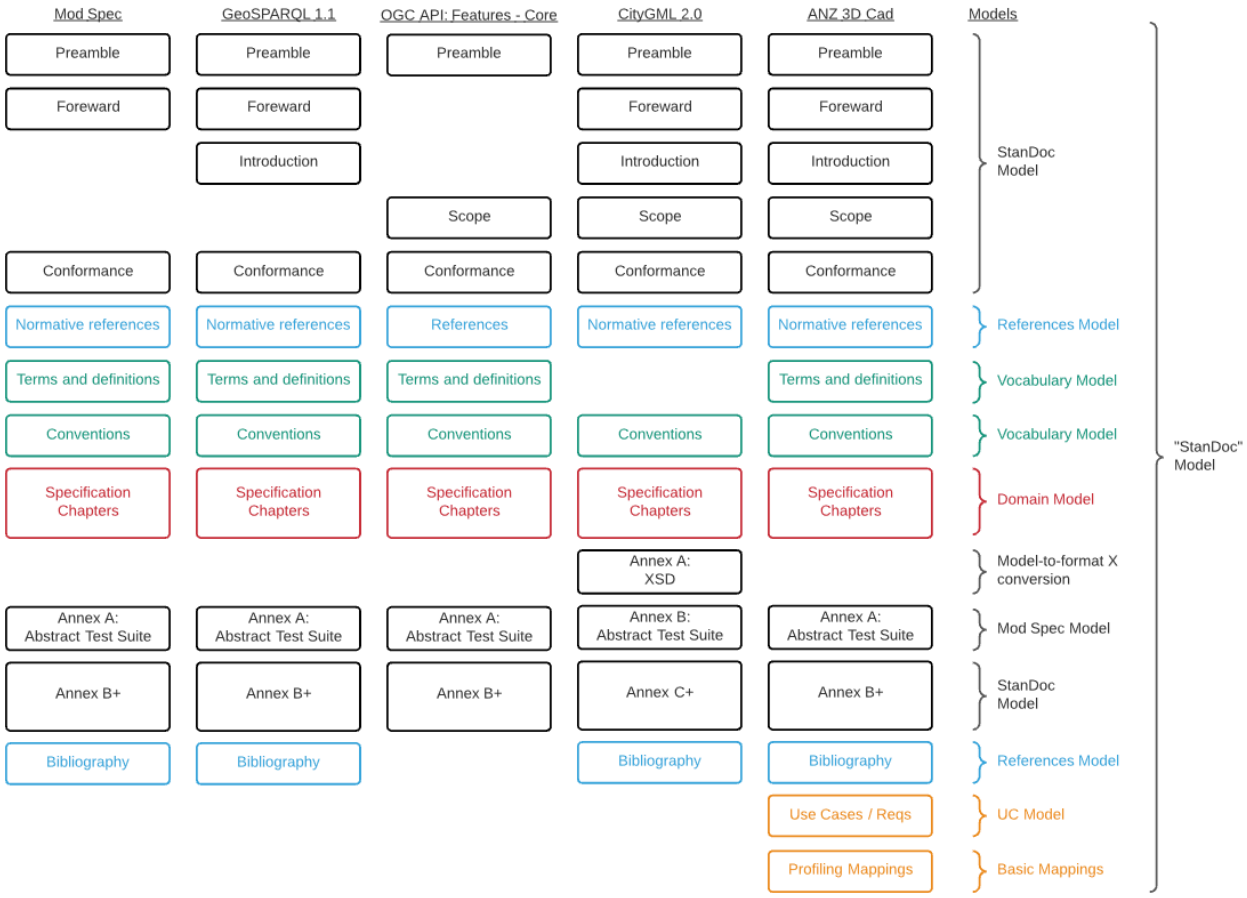


Figure 62 — Parts of the Specifications documents of several Standards

9.3.7. Specification part models

9.3.7.1. Document Structure

A so-called *StanDoc* ("Standards Document") model is indicated in the figure above to model the overall structure of Specifications and several parts within them. SURROUND has used the Semantic Publishing and Referencing Ontologies (SPAR Ontologies)[3] to model structured documents such as Standard thus SPAR Ontologies are proposed as a candidate StanDoc model.[4]

[3] http://www.sparontologies.net/

[4] This RDF-based StanDoc model differs from the Metanorma Standard Document (StanDoc) model, a conceptual model and associated Relax NG XML Schema used to express standards documents by Clause 3.10.5, and documented in ISO/AWI 36100. While the two models appear to address an overlapping domain, the Metanorma StanDoc model cover exhaustive details and has been adopted by over 15 SDOs.

SPAR is a set of related but orthogonal models originally designed for academic and technical publication structural and semantic modelling. Simple SPAR modelling describes elements familiar to all structured document producers: *Section*, *Table of Content*, *Title* etc.

Proposing the use of a semantic model for document structure is a more formal way of defining Standard content than the use of a set of templates or descriptions of document elements, as the OGC currently uses. Ontology *Shapes* may be defined in a profile of SPAR tuned to the OGC's Standard's needs. This profile, which has not been created in this Testbed, could be made to meet current OGC requirements and even map to non-Semantic document systems, such as ASCIIDOC templating.

A major benefit to using a semantic document structural model is that with such a model implemented, Specifications' structural elements become data similarly to their Domain Model and other parts. Use then of ontologies for document structure is the ultimate Model-Driven Standard step. Below is shown an extract of the ANZ 3D Cad Standard's Specification's SPAR modelling. The content follows the commonly seen OGC Specification structure.

```
<https://example.com/anz-3d-cad>
  a frbr:Work , owl:Ontology ;
  co:firstItem _:b626767 ;
  co:item [
      a co:ListItem ;
      co:index 1 ;
      co:itemContent _:b626767 ;
    ] ;
  ...
  co:item [
      a co:ListItem ;
      co:index 9 ;
      co:itemContent _:b865408 ;
    ] ;
  po:contains :preamble ;
  po:contains :scope ;
  po:contains :related-domains ;
  ...
  po:contains :annex-b ;
  po:contains :annex-c ;
  dcterms:published "2021-11-23"^^xsd:date ;
  dcterms:title "ICSM Conceptual Model for 3D Cadastral Survey Dataset
Submissions"@en ;
  frbr:embodiment [
      a prof:ResourceDescriptor ;
      dcterms:format "text/html" ;
      rdfs:comment "OGC-style Specification online"@en ;
      rdfs:label "PDF of the ICSM Conceptual Model for 3D Cadastral Survey
Dataset Submission"@en ;
      prof:hasArtifact "http://cad-spec.s3-website-ap-southeast-2.amazonaws.
com/"^^xsd:anyURI ;
    ] ;
.
```

Figure 63 — ANZ 3D Cad Standard's Specification's SPAR modelling

9.3.7.1.1. Terms & Definitions Model

Specifications often contain *Terms & Definitions* and sometimes *Conventions* sections. Both of these types of sections are easily modelled using a semantic vocabulary model such as W3C TR skos-reference. Vocabularies of terms & definitions are both critical for Specifications, easy to model in something like SKOS but hard to model well in *Domain Model* systems such as UML Class models.

Double modelling can take place for these sections where defined terms and conventions are both modelled as document structural elements using SPAR Ontologies and also as vocabulary concepts using SKOS which allow for vocabulary content use in isolation — outside a document. Having two forms of modelling, side-by-side, is entirely possible in SW modelling and specifications such as *Content Negotiation by Profile* (W3C TR dx-connegp) indicate how to indicate when one or other form of content's modelling should be used.

An intention of the OGC Naming Authority (OGC-NA) is to present a set of definitions, via the OGC Definitions Server[5], that both includes all the defined *Terms & Definitions* and *Conventions* from OGC Specifications and makes them available for reuse. The OGC Definitions Server is a part of the infrastructure that provides the OGC Body of Knowledge. So, if a Specification needed to contain a definition for the term *Boundary*, it would contain the ASCIIDOC content:

```
==== Boundary

Set that represents the limit of an entity.

NOTE: Boundary is most commonly used in the context of geometry, where the
set is a collection of points or a collection of objects that represent those
points. In other arenas, the term is used metaphorically to describe the
transition between an entity and the rest of its domain of discourse

[.source]
<<ISO_19107_2003,clause=4.4>>
```

Figure 64 — ASCIIDOC representation of term *Boundary*

It could, instead, use the SKOS Concept from the OGC Definitions Server at https://www.opengis.net/def/CaLAThe/4.0/Boundary:

```
@prefix skos: <http://www.w3.org/2004/02/skos/core=> .

<https://www.opengis.net/def/CaLAThe/4.0/Boundary>
    a skos:Concept ;
    skos:broader <https://www.opengis.net/def/CaLAThe/4.0/LandDivision> ;
    skos:definition "Set that represents the limit of an entity. Note 1 to
entry: Boundary
        is most commonly used in the context of geometry, where the set is a
collection of
```

[5] http://defs.opengis.net/vocprez/

```
        points or a collection of objects that represent those points. In
other arenas, the
        term is used metaphorically to describe the transition between an
entity and the
        rest of its domain of discourse."@en ;
    dcterms:source "https://www.iso.org/standard/26012.html"^^xsd:anyURI ;
    skos:narrower <https://www.opengis.net/def/CaLAThe/4.0/BoundaryMark> ;
    skos:prefLabel "Boundary"@en ;
.
```

Figure 65 — SKOS representation of term *Boundary*

That SKOS RDF code could easily be converted to the equivalent ASCIIDOC code for display in a document form of the Specification but the use of a Concept in the OGC Definitions Server makes the defined term far more machine-readable and more accessible and traceable generally.

OGC Specification' *Terms & Definitions* and *Conventions* elements usually support *Domain Model* elements which can be said to depend upon their definitions. Where the *T&D* & *Convention* elements, in turn, depend on Naming Authority or other defined terms, we may have relations as per Figure 66 below.

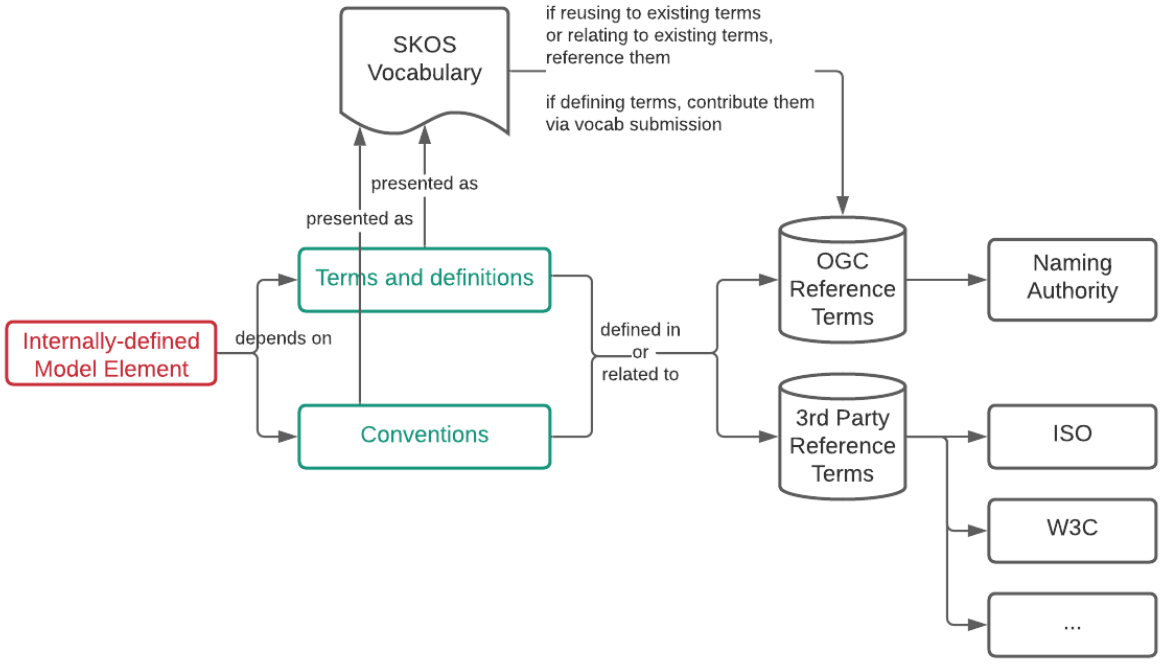


Figure 66 — Conceptual relations between Domain Model elements ("Internally-Defined Model Elements") and OGC and external reference terms

This element of Model Driven Standards moves from driving a particular Standard's section — *Terms & Definitions* within a *Specification* from being driven by a local model, i.e. for this Standard only, to the section being driven by a combination of new and previously defined elements from a larger Standards' baseline set of models.

9.3.7.1.2. References Model

Most OGC (and ISO and W3C) Specifications contain normative and non-normative reference sections, as shown in Figure 62. The SPAR Ontologies contain a detailed handing of reference modelling. The same SPAR reference handing can also be used for Specification's Bibliographies.

Figure 67 is a conceptual diagram of how different parts of a Specification reference and define *Terms & Definitions*, *Normative References* and *Bibliography* elements.

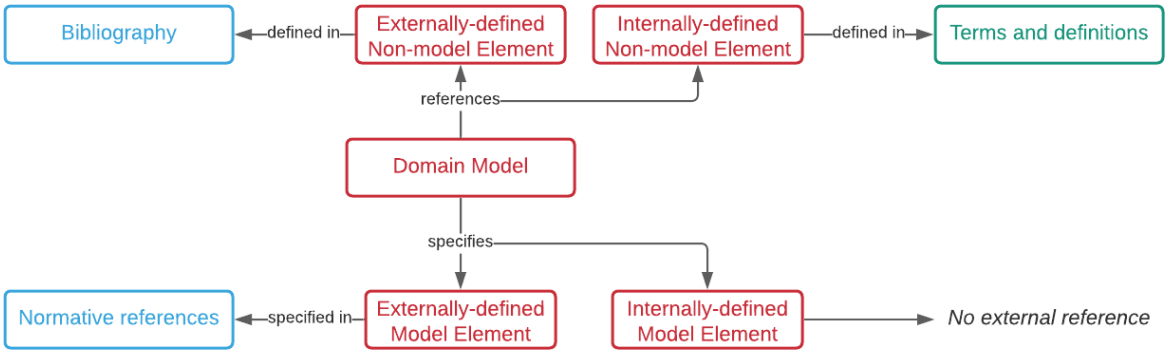


Figure 67 — Conceptual modelling of how a Specification's *Domain Model* elements relate to *Terms & Definitions*, *Normative References* and *Bibliography* section elements.

Modelling such as this, along with its formal realisation in SPAR Ontologies, would provide machine-readable linking between OGC Specifications elements that have previously not been linked in this way: currently OGC Specifications do not link *Domain Model* elements to other definitional Specification elements. Use of reference modelling in this way will expand the envelope of the model driven elements of a Standard.

9.3.7.1.3. Test Suite (Requirements) Model

Many OGC Specifications contain an Abstract Test Suite and there are many OGC Specifications that contain strictly structured Abstract Test Suite elements, for example the *OGC API — Features — Part 1: Core* standard OGC 17-069r3. *OGC API — Features* defines "Conformance Classes" and "Abstract Tests", all identified with URIs with relationships — the Test Suite / Conformance model — taken from the OGC ModSpec (OGC 08-131r3). The model is presented in UML and could easily be presented in OWL. This has not been done in this Testbed, however the GeoSPARQL 1.1 release contains not only "Conformance Classes" and "Abstract Tests" presented in a similar way to *Features*, but also contains a SKOS vocabulary of the items too (`reqs.ttl`), so it is close to this modelling. It would require little effort to build the sorts of Requirements / Tests listing seen in OGC specifications from this vocabulary or from a more semantically strong OWL version of this content.

One example of a Requirement and its related Conformance Test from GeoSPARQL 1.1's vocabulary of them is given in the listing below:

```
reqs10tve:rcc8-spatial-relations
    a spec:Requirement ;
    skos:prefLabel "GeoSPARQL 1.0 Requirement: RCC8 Spatial Relations"@en ;
    skos:definition "Implementations shall allow the properties geo:rcc8eq,
geo:rcc8dc, geo:rcc8ec, geo:rcc8po, geo:rcc8tppi, geo:rcc8tpp, geo:rcc8ntpp,
geo:rcc8ntppi to be used in SPARQL graph patterns" ;
.

conf10tve:rcc8-spatial-relations
    a spec:ConformanceTest ;
    skos:prefLabel "GeoSPARQL 1.0 Conformance Test: RCC8 Spatial Relations"@en
;
    skos:definition "Verify that queries involving these properties return the
correct result for a test dataset" ;
    spec:testPurpose "check conformance with this requirement" ;
    spec:testType spec:Capabilities ;
    rdfs:seeAlso reqs10tve:rcc8-spatial-relations ;
.
```

Figure 68 — GeoSPARQL 1.1 Requirement and its related Conformance Test

While above one Requirement is indicated by one Conformance test, other Specifications could contain non-1:1 relations.

GeoSPARQL 1.1 also groups Requirements so that *Conformance Classes* are created as collections of Conformance Tests. The tests for the GeoSPARQL 1.1 "core" *Conformance Class* are listed below:

```
confs11:core
    a spec:ConformanceClass, skos:Collection ;
    skos:member conf11core:feature-collection-class ,
                conf11core:spatial-object-class ,
                conf11core:sparql-protocol ,
                conf11core:spatial-object-collection-class ,
                conf11core:spatial-object-properties ;
    skos:prefLabel "GeoSPARQL 1.1 Conformance Class: Core"@en ;
.
```

Figure 69 — GeoSPARQL 1.1 core Conformance Class

Most of the Conformance Test / Class and Requirements modelling is just straightforward SKOS-based vocabulary modelling of concepts with a few special properties to indicate tests purpose (`spec:testPurpose`) and similar.

9.3.7.2. Domain Model

9.3.7.2.1. UML to OWL

Domain models within the OGC often reference a "baseline" of ISO models, such as ISO 19107:2019 and GeoSPARQL 1.1, and the ANZ 3D Cad models are no different: GeoSPARQL's *Standard Structure* introductory text describes its use of these models[6].

The ISO's TC 211 presents not only the consolidated UML versions of these models as a "harmonized model", but also a consolidated OWL version of them too via ISO/TC 211's *Group on Ontology Maintenance* (GOM)'s public code repository (ISO/TC 211/GOM Repository). While this consolidated OWL information is subject to many questions regarding its exhibition of *Ontology Design Patterns*[7] or, generally, "Best Practices", a UML to OWL conversion for multiple (ISO) standard's Domain Models is demonstrated.

There are many UML-to-OWL academic papers and studies, for example Brockmans et al. 2004 and Jetlund et al. 2019.

The first publication was made early in the days of OWL modelling but remains relevant. More recent, simple UML-to-OWL mappings/models exist[8], but the fundamentals remain unchanged. These sorts of mappings are the types used to generate GOM Semantic Web data.

The second publication uses essentially current ISO TC 211 Standards' UML (circa 2019) and concludes that "conversion challenges [from UML to OWL] are addressed by adding more semantics to UML models: global properties and reuse of external concepts". This author agrees with the particular solution of that paper and feels that a next-generation of automatically-produced OWL from the current TC 211 UML could be made that exhibits many more Ontology Design Patterns.

9.3.7.2.2. OWL to UML

The GeoSPARQL (1.0 and 1.1) and ANZ 3D Cad standards have Semantic Web-first models, that is they have models presented initially as OWL ontologies and then, informally, UML diagram versions of them, or parts of them. GeoSPARQL 1.1's main Domain Model diagram, from its

[6]https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html=_geosparql_standard_structure

[7]Whole research communities are deveoted to Ontology Design Patterns and there are listings of such patterns online such as http://ontologydesignpatterns.org/wiki/Main_Page. No formal rating of the GOM's ontologies has been conducted with respect to these, however, internally, the GOM has been able to show that some patterns are violated and the intention for GOM's operations in 2022/2023 is to show explicity violations and then to address them.

[8]For a modern, succinct example, see https://henrietteharmse.com/uml-vs-owl/uml-class-diagram-to-owl-and-sroiq-reference/

Specification document, is presented below in Figure 70: it is not a UML diagram but an informal OWL diagram (See Clause 9.5.3 for information on this type of diagramming).

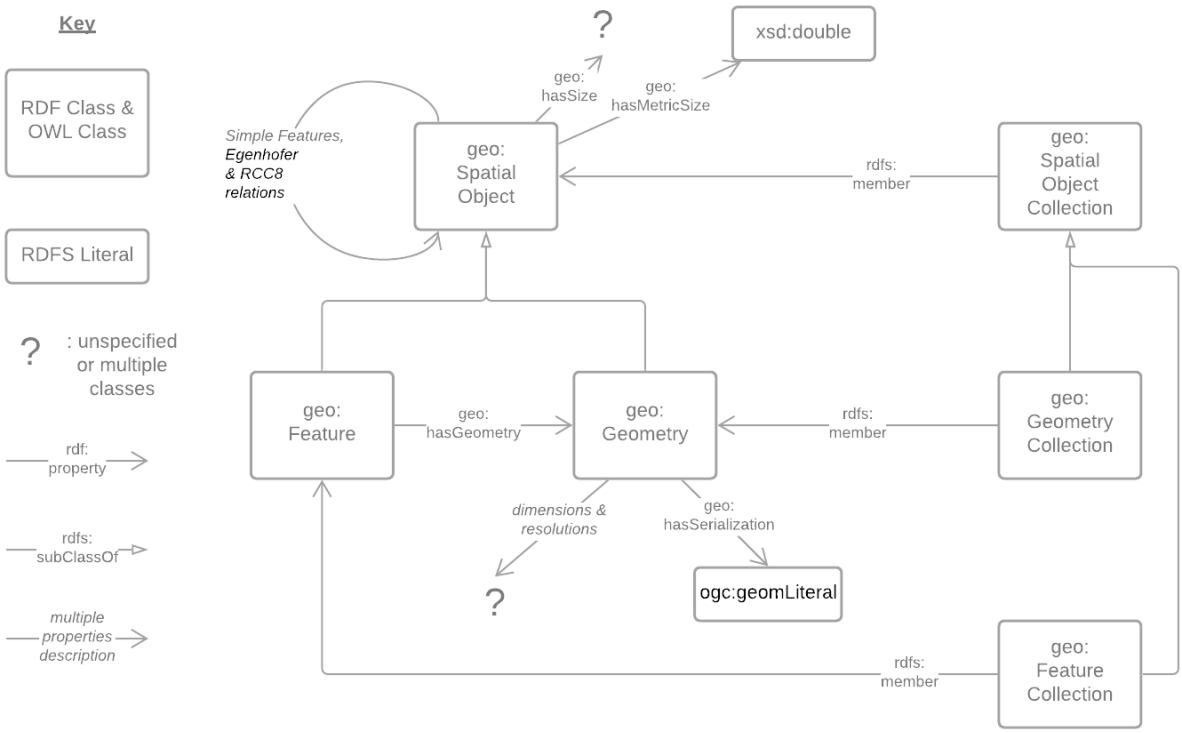


Figure 70 — An informal model diagram overview of GeoSPARQL 1.1's Domain Model. After https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html=_core

A simple UML diagram of a model presented at the level of detail of the figure above is easy to make, and such diagrams are presented as the main way to communicate OWL model structures for some well-known ontologies (see Clause 9.5.5 for examples). For work within this testbed, basic UML diagramming was easily generated for a couple of modelling scenarios, see Clause 9.5.5.

Tools that can automatically generate UML diagrams from OWL include:

- OWLGrEd — http://owlgred.lumii.lv/
- TopBraid Composer ontology editor — https://www.topquadrant.com/composer/featureClassDiagram.html
- Protege ontology editor, via plugins — https://protegewiki.stanford.edu/wiki/OWL2UML

It is likely, but unproved, that the UML diagrams for Domain Models similar to those in the ISO's Technical Committee 211's remit could be automatically produced from either GOM OWL ontologies or from slight variants of them.

Unlike UML-native objects, objects modelled initially in OWL but then expressed in UML would retain the original richer semantics of the OWL form, allowing for better integration with other Semantic Web-based technologies such as SHACL-based (W3C TR shacl) data validators and

SKOS-based (W3C TR skos-reference) vocabulary definitions. Following the reasoning of Jetlund et al. 2019 regarding specifying a profile of UML to form OWL, a profile of OWL could easily be made that forms UML of the sort required by OGC or ISO standards. Such a profile could be implemented using W3C TR dx-prof.

9.3.7.2.3. Domain modelling

Regarding the specific modelling required for OGC/ISO standards: the ISO standard's "baseline", as referenced above, creates OWL classes of types recognizable to users of the UML models versions of their standards, given that the OWL classes are auto-generated from the UML. For instance, ISO 19160 (Addressing)'s OWL version (ISO 19160-1:2015) presents an `Address` class[9] and references classes from more fundamental models, such as ISO 19107's `GM_Object` (ISO 19107:2019).[10]

It is straightforward to extend this baseline for other Standards, and where the examples in Clause 9.5 show GeoSPARQL's `geo:Feature` as a more abstract class (superclass) for their specific classes, links to ISO 19107's `GM_Object` exist, given that GeoSPARQL indicates `geo:Feature` is a subclass of `GM_Object`[11].

9.3.7.3. Other Models

Several elements not often seen in existing OGC Specifications are present in the ANZ 3D Cadastre Standard's Specification. Most prominent is a Use Cases Annex whose content is modelled with a simple Use Case Ontology created for this purpose.[12]

Use Case modelling is well-known in UML but, as far as the author is aware, no Specifications other than the ANZ 3D Cadastre Standard's model Use Cases in such a way that Domain Model elements may be linked to the Use Cases that motivated their implementation.

While it would be possible to implement a UML-based, non-semantic, Use Case model within a Specification, use of a Use Case ontology within the SW envelope of models allows for Domain Model / Use Case model element referencing using SPAR Ontology referencing, as per Clause 9.3.7.1.2. Figure 71 shows a conceptual linking for this.

[9] https://github.com/ISO-TC211/GOM/blob/master/isotc211_GOM_harmonizedOntology/iso19160/-1/2015/Address.rdf=L77

[10] https://github.com/ISO-TC211/GOM/blob/master/isotc211_GOM_harmonizedOntology/iso19160/-1/2015/Address.rdf=L1261 and many other places

[11] Note that when GeosPARQL 1.0 was created (2012), there was no TC 211 ontology expression of `GM_Object` and thus no OWL link to one is present in GeoSPARQL.

[12] https://data.surroundaustralia.com/def/uc

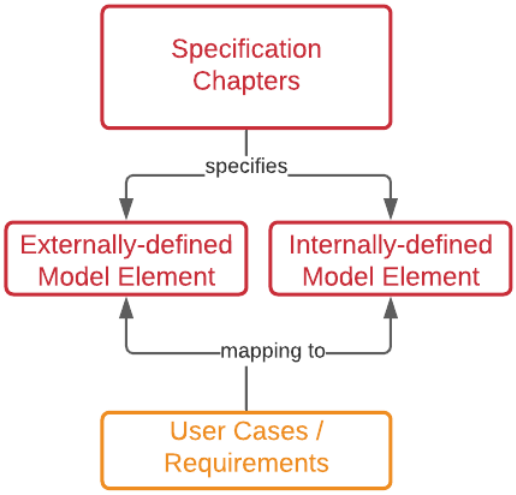


Figure 71 — Conceptual modelling of how a Specification's *Domain Model* elements relate to *Use Case / Requirements* section elements.

9.3.8. Implementation Examples Modelling

The ANZ 3D Cad Standard presents selectable variants of its Specification's content for different jurisdictional users. The jurisdictions all have different model terminology and implementation patterns, but the different elements have been mapped to the Standard's core conceptual "canonical" model. A jurisdictional user of the Specification is able to turn on and off individual jurisdictional elements in the document to see only the forms they wish to. To facilitate this, the ANZ 3D Cad Specification contains multiple, different, *Implementation Examples* of the Standard's Domain Model for the jurisdictions. While it would be possible to simply link multiple Implementation Examples to Domain Model elements and then to tag individual ones as being relevant for one or more jurisdictions, the Specification actually contains mapping from Domain Model elements to Implementation Example objects which are then described with "Resource Descriptor" properties. Such properties, taken from the *Profiles Vocabulary* (W3C TR dx-prof), include `dcterms:conformsTo` which allows for the indication of Standards to which the Implementation Example object conforms. Clearly the Implementation Example must conform to the concepts in the Domain Model (and they all do), but then they also conform to other syntactic or diagrammatic or schema models, such as JSON, UML Package Diagram or others that then allow users to understand the *form* of the Implementation Example — what language, schema, diagram etc. it uses to implement the Domain Model.

In Figure 72 and Figure 73 below, the `AdoptedVector` and `LandPropertyUnit` classes of the ANZ 3D Cad Standard's Domain Model are shown with Implementation Examples that conform to them and which also conform to other specifications — *LandXML* and *ClassContextDiagram* respectively.

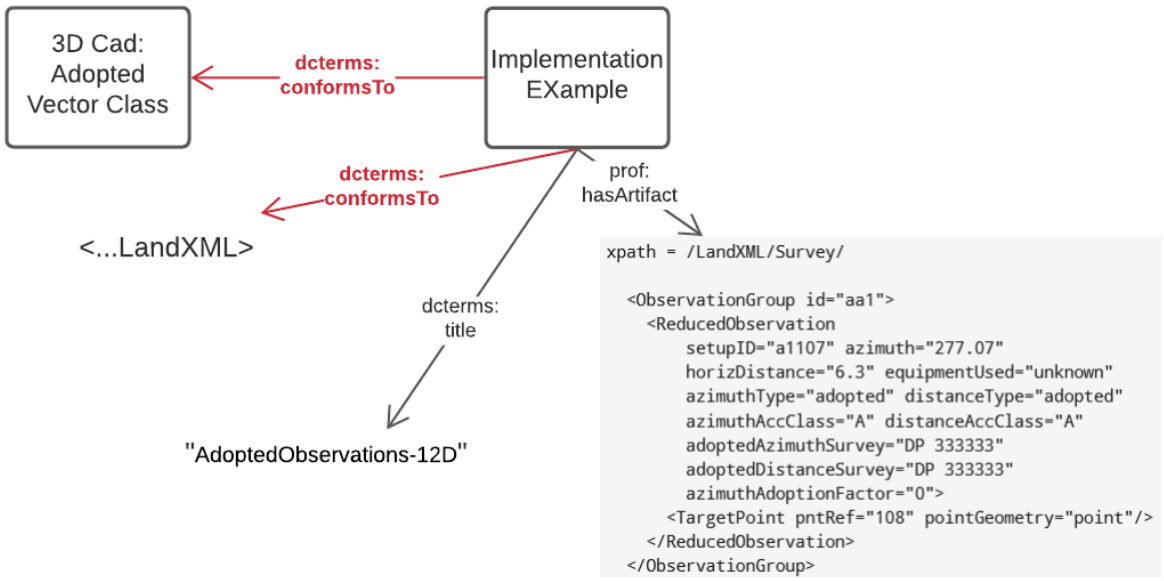


Figure 72 — The ANZ 3D Cad model's `AdoptedVector` class with a LandXML Implementation Example

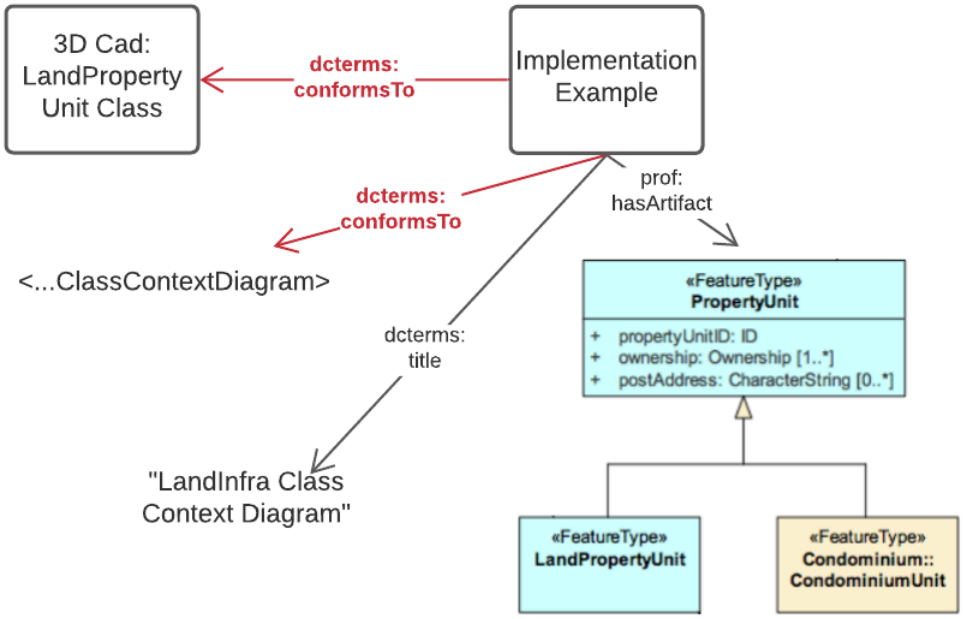


Figure 73 — The ANZ 3D Cad model's `LandPropertyUnit` class with a documentation diagram conforming to a *ClassContextDiagram* Standard

Differentiating types of conformance are not possible in the *Profiles Vocabulary* (W3C TR dx-prof), thus no distinction is made between the way in which the Implementation Examples in the figures above conform to elements in the Domain Model and the other Standards that they

conform to. How to differentiate types of conformance, and whether such a thing is sensible, has simply not been determined.

9.3.9. GeoSPARQL Standard Example

The GeoSPARQL 1.1 Specification is online at:

- https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html

It is auto-built from ASCIIDOC source files located at https://github.com/opengeospatial/ogc-geosparql/tree/master/1.1/spec using standard ASCIIDOC tooling. While it is conceptually a Model Driven Standard in that many parts of the Specification are backed by technical models, such as the Domain Model, Terms & Definitions vocabulary, Requirements listing etc., it is not auto-Model Driven.

It is planned that GeoSPARQL 1.2, to be created in 2022, will be auto-Model Driven with the document structure and many elements being auto built from data files (OWL ontologies).

The full listing of GeoSPARQL 1.1 elements, including the Specification document and the technical artifacts for the Terms & Definitions sections, etc., are listed in the GeoSPARQL 1.1 Profile Declaration at:

- https://opengeospatial.github.io/ogc-geosparql/geosparql11/profile.html

Note that this declaration is also specified in machine-readable form (RDF data, according to W3C TR dx-prof) at https://opengeospatial.github.io/ogc-geosparql/geosparql11/profile.ttl.

The automated build process for the Specification's ASCIIDOC to HTML conversion and for the generation of other documentation, such as the Profile Declaration, is specified in GeoSPARQL 1.1's infra-coding scripts online at https://github.com/opengeospatial/ogc-geosparql/tree/master/.github/workflows.

9.3.10. ANZ 3D Cadastre Standard Example

The ANZ 3D Cadastre Specification is online in several forms at:

- http://cad.surroundaustralia.com/

It is auto-built from data files (OWL ontologies) with tooling converting OWL ontologies to ASCIIDOC intermediate files and then converting those to HTML files. This process is more fully detailed in Clause 9.4.

9.4. oMDS Specification Production

oMDS is an acronym for the *(ontology) Model-Driven Standards* tool used to create the ANZ 3D Cadastre Specification.

9.4.1. Specification variants

Several forms of the ANZ 3D Cadastre Specification are produced from model data and are online at:

- http://cad.surroundaustralia.com/

The multiple versions here are:

- a traditional Specification document
  - with normative/non-normative element selectors
  - with profile content selectors
- Specification document with embedded feedback forms
  - to allow non-technical readers of the Specification to supply feedback, per Specification element
- Specification with experimental elements
  - this version contains a 3D data viewer and other ontilie functionality extensions on top of what is usually see within a Specification

9.4.2. Build sequence

All versions of the Specification are built in a similar way, which is:

1. OWL Ontology content stored in a database is read by the oMDS tool and converted to in-memory Python objects
   - each part of the ANZ 3D Cad model is stored in a separate *graph*, that is an isolated part of the overall ontology used. Graphs here are akin to the SQL relational database notion of a *schema*

2. In-memory objects are serialized to ASCIIDOC of a similar form to most OGC Standards, including GeoSPARQL 1.1

    - mapping between ontology models for the Specifications various sections, e.g. Use Cases, Domain Model, Terms & Definitions, are currently retained within the oMDS tool but may be externalized, for potential wider use, later
    - mappings take the form of Python classes, instances of which are generated by querying the ontologies using SPARQL queries and then serialized as needed by calling a `to_asciidoc()` function that templates Python object instance variables into ASCIIDOC

3. ASCIIDOC is converted to HTML or PDF using the standard of ASCIIDOCtor tool

    - the next generation of the oMDS tool will incorporate a Python-based ASCIIDOC to HTML converter, rather than relying on the external ASCIIDOCtor tool

4. Build parameters determine which, of several, HTML post-processing steps are run on the ASCIIDOCtor-generated HTML

    - these post processing steps allow for the inclusion of functionality beyond the standard ASCIIDOC offering for features such as:
        - normative/non-normative element selectors, jurisdictional profile selectors, embedded feedback forms, element highlighting (for incomplete development stages) etc.

5. Infracode tools place the auto-generated Specification documents online

    - at http://cad.surroundaustralia.com/

Much of the OWL-to-ASCIIDOC conversion approach was derived from the open source OWL Ontology documentation tool RDFLib pyLODE.

9.4.3. Build element isolation

Each element of the oMDS-generated Specification can be built in isolation or in an ensemble so that, during testing, a single part may be worked on. This is partly enabled by the ability for ASCIIDOC documents to be generated either in isolation or via compilation through import statements. It is also partly enabled by the isolation of conceptual models for different parts, for example, the Terms & Definitions section, while referencing elements in other sections, forms a complete SKOS vocabulary by itself. Likewise, the Use Case information in the Specification forms complete instances of the SURROUND Use Case Ontology[12] even though the Use Case elements (Requirements etc.) are referenced by elements in the Domain Model.

9.4.4. oMDS next steps

Currently the oMDS tool is purpose-built for the ANZ 3D Cad Specification, however it has always been intended that the tool will be generalized for use with other (ontology) Model-Driven Standards. The following development specifications are executed for the oMDS tool in the first half of 2022 for at least the auto-generation of the GeoSPARQL 1.2 Specification and perhaps other Specifications too:

- ensure that *all* ASCIIDOC content is produced from OWL data using in-memory Python objects only
    - currently this is mostly the case, however some Python ASCIIDOC templating is still also used
- use of Python ASCIIDOC to HTML conversion
    - to 'inline' this conversion step, rather than rely on an external (to the oMDS tool) converter
- extraction of ASCIIDOC-generated CSS into separate resources from the output HTML files
    - for browser caching
- position the Specification elements according to a SPAR Ontologies document model
    - while the ANZ 3D Cad specification is modelled according to SPAR Ontologies, it is currently still produced from ordered ASCIIDOC documents which are themselves produced from models

9.5. OWL Diagramming

9.5.1. General

Early within Testbed 17, there was much discussion about the role of the visual element of UML diagrams in the creation of OGC Standards. One set of thoughts related to the formality of the diagrams and how they did, or didn't, correspond to formal UML models. The general direction sought, going forward in the OGC, was that while UML diagrams are useful or perhaps even necessary as visual artifacts, they either should, or even must, be based on formal UML models that standards editors must supply in addition to the visual representation of models to allow for

model-driven standards. This set of thoughts is detailed most completely in the OGC Testbed-17 D023 UML Modeling Best Practices.

A parallel discussion was then entered into regarding the potential for visual diagramming in OWL, with some thoughts being that OWL modelling would need to have some form of equivalence to UML diagramming for future Standards editors to be able to even consider it in place of UML.

This section relates a series of investigations into OWL Diagramming within the Testbed activities.

9.5.2. Diagram styles

OWL diagramming is less well known and not as easy to overview as UML diagramming principally because OWL is not first and foremost a visual design tool, as UML is, or at least was. OWL is a modelling system that is both system-independent and system-implementable. There are a series of OWL diagramming *styles* but no widely accepted *official* or *correct* form of OWL diagramming. The next section and subsections list several forms of OWL diagramming that were demonstrated during Testbed 17 with pros and cons, as they relate to OGC Standards, given.

An example piece of an OWL model, replicating CityGML's Tunnel class and its relations, is used for the next few diagram styles. The OWL code of that example, in the W3C TR turtle syntax, is given below. Snippets of that code will be explained in other code blocks, as needed.

```
@prefix cgml: <http://example.com/citygml/> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

cgml:HollowSpace a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasHollowSpace ] ;
            owl:someValuesFrom owl:Thing ],
        geo:Feature .

cgml:Tunnel a owl:Class,
        owl:Ontology ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:isTunnelPartOf ] ;
            owl:someValuesFrom owl:Thing ],
        cgml:TunnelPart .

cgml:TunnelConstructiveElement a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasTunnelConstructiveElement ]
;
            owl:someValuesFrom owl:Thing ] .

cgml:TunnelFurniture a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasTunnelFurniture ] ;
```

```
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasClass ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasFunction ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasUsage ;
            owl:someValuesFrom owl:Thing ] .

cgml:TunnelFurnitureUsage a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasUsage ] ;
            owl:someValuesFrom owl:Thing ],
        skos:Concept .

cgml:TunnelInstallation a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasTunnelInstallation ] ;
            owl:someValuesFrom owl:Thing ] .

cgml:TunnelPart a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty cgml:hasClass ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasHollowSpace ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasTunnelConstructiveElement ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasTunnelFurniture ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:hasTunnelInstallation ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty cgml:isTunnelPartOf ;
            owl:someValuesFrom owl:Thing ],
        geo:Feature .

dcterms:isPartOf a owl:ObjectProperty ;
    rdfs:label "is part of" .

skos:Concept a owl:Class ;
    rdfs:subClassOf [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasClass ] ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasFunction ] ;
            owl:someValuesFrom owl:Thing ],
        [ a owl:Restriction ;
            owl:onProperty [ owl:inverseOf cgml:hasUsage ] ;
            owl:someValuesFrom owl:Thing ] .

cgml:hasFunction a owl:ObjectProperty .

cgml:hasHollowSpace a owl:ObjectProperty .

cgml:hasTunnelConstructiveElement a owl:ObjectProperty .
```

```
cgml:hasTunnelFurniture a owl:ObjectProperty .

cgml:hasTunnelInstallation a owl:ObjectProperty .

cgml:isTunnelPartOf a owl:ObjectProperty ;
    rdfs:subPropertyOf dcterms:isPartOf .

geo:Feature a owl:Class .

cgml:hasClass a owl:ObjectProperty .

cgml:hasUsage a owl:ObjectProperty .
```

Figure 74 — OWL modelling of CityGML Tunnel class

9.5.3. Informal OWL diagrams

Many OWL ontologies, the equivalent of a UML domain model, have informal diagrams, that is diagrams following no strict specification. Such diagrams are common to see and appear in well-respected ontology documents such as:

- Epimorphics' *Registry Ontology*
  - https://epimorphics.com/public/vocabulary/Registry.html
- GeoSPARQL 1.1
  - diagram by this author
  - https://opengeospatial.github.io/ogc-geosparql/geosparql11/spec.html

Table 6 — Pros and Cons of informal OWL diagrams

PROS	CONS
Simple	Not 1:1 with technical model
Able to convey points of interest	

NOTE: Even though informal OWL diagrams are not 1:1 with technical models, it's not possible, or at least it's not expected, that any OWL modeller could characterise an OWL model without a technical artifact, so there either cannot be, or at least there so far has never been, an instance of a specification that has an informal OWL diagram and no resource that can be used for model-driven standards work.

Here is an informal diagram showing most of the elements of the CityGML `Tunnel` model:

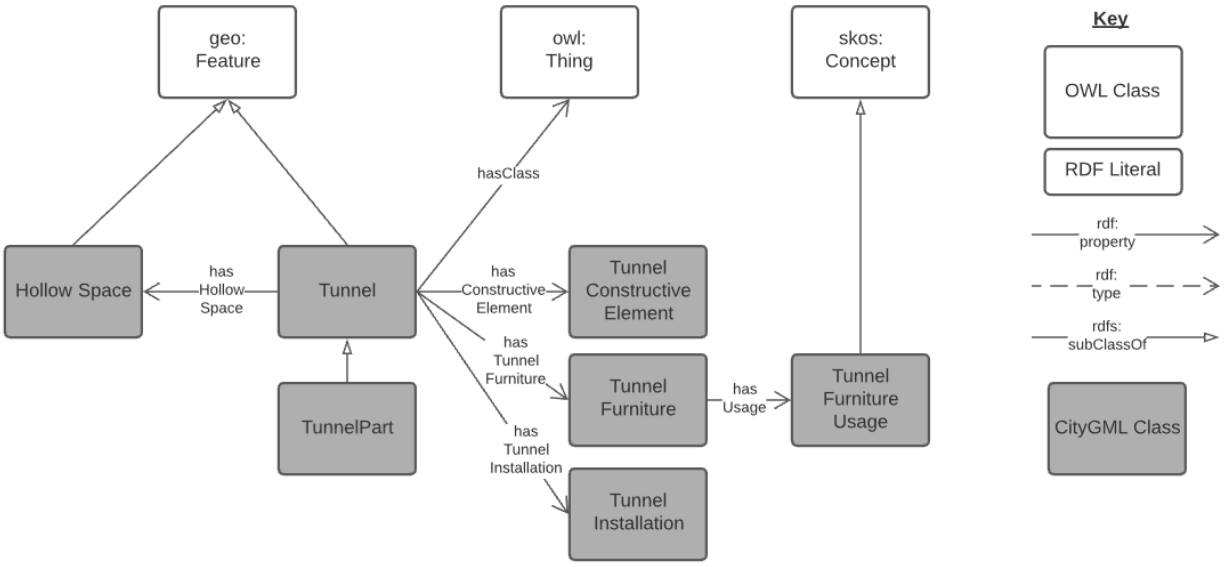


Figure 75 — Part of CityGML's *Domain Model* focused on the `Tunnel` class represented with an informal OWL diagram. Note the use of some special property/association arrow styles. Other properties/associations have their type indicated on the instance

9.5.4. Informal OWL diagram tooling

The above diagram was drawn by hand using a generic diagramming tool. Many ontology editing systems provide auto informal diagramming, for example the well-known Protégé ontology editor contains a plugin called *OntoGraph* that produces diagrams like the following from the same CityGML content used above:

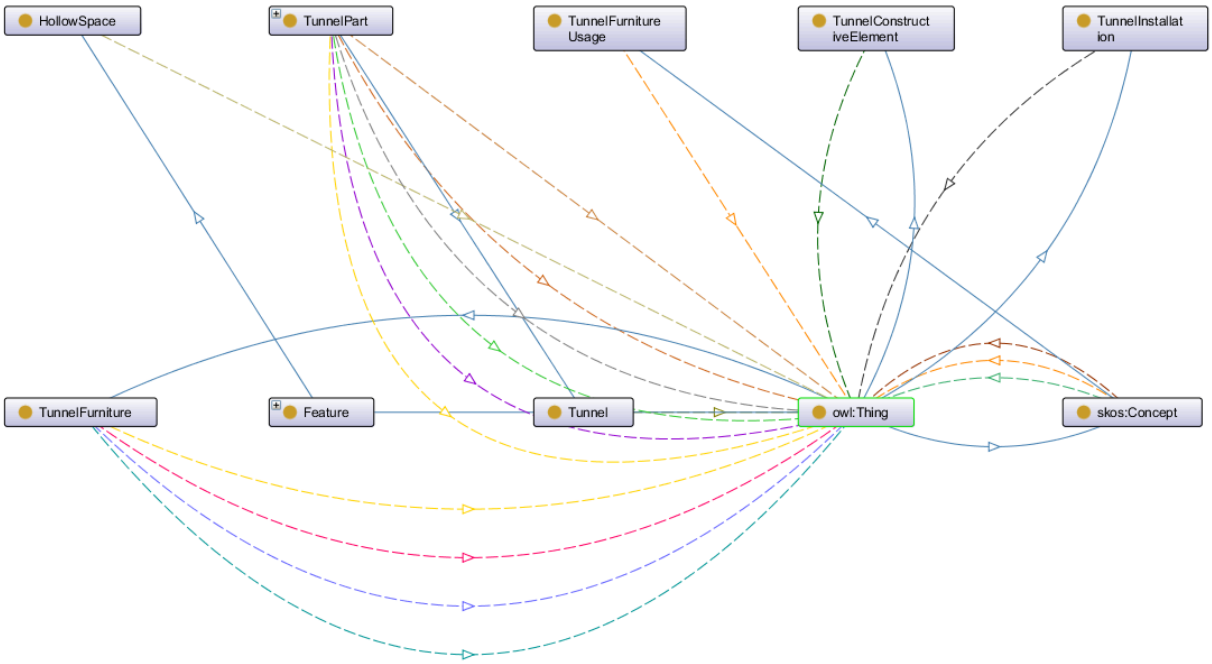


Figure 76 — Informal OWL drawing of CityGML's `Tunnel` class made by Protégé's *OntoGraph* plugin

The relationships shown in the diagram above are typed and able to be shown or hidden using selectors built into Protégé, such as in the figure below:

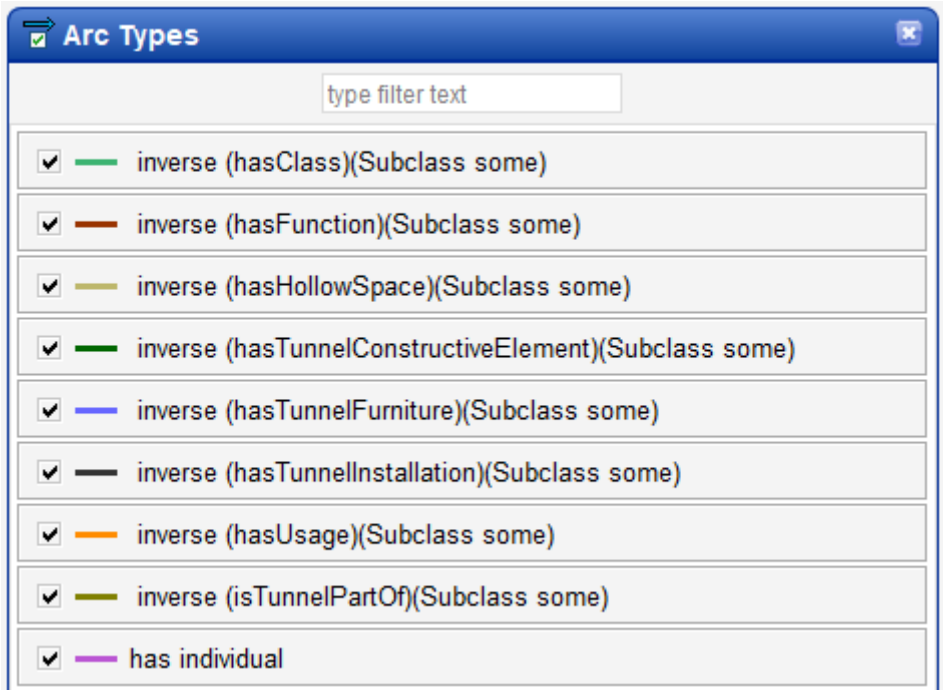


Figure 77 — Protégé's *OntoGraph* plugin's relationships dialogue

OntoGraph images, which can also be created outside Protégé, can be rearranged automatically or by hand. The following two images show firstly an auto-layout, 'hierarchical' and a layout specified by hand.

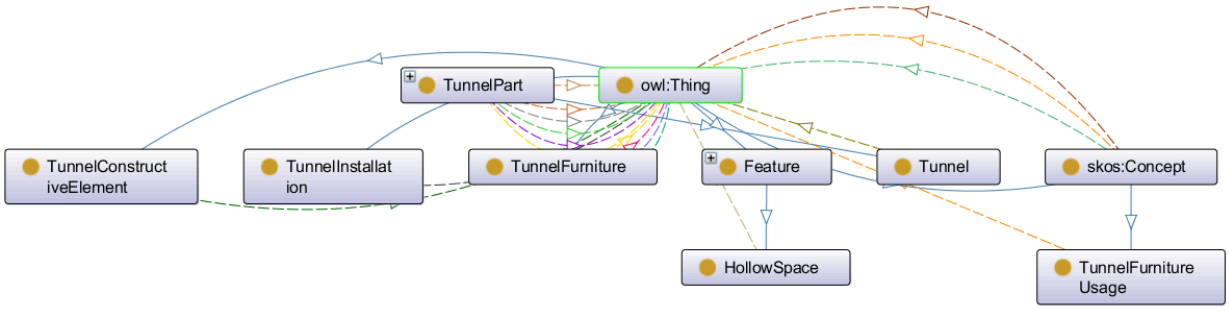


Figure 78 — Informal OWL drawing of CityGML's `Tunnel` class made by Protégé's *OntoGraph* plugin, hierarchical layout

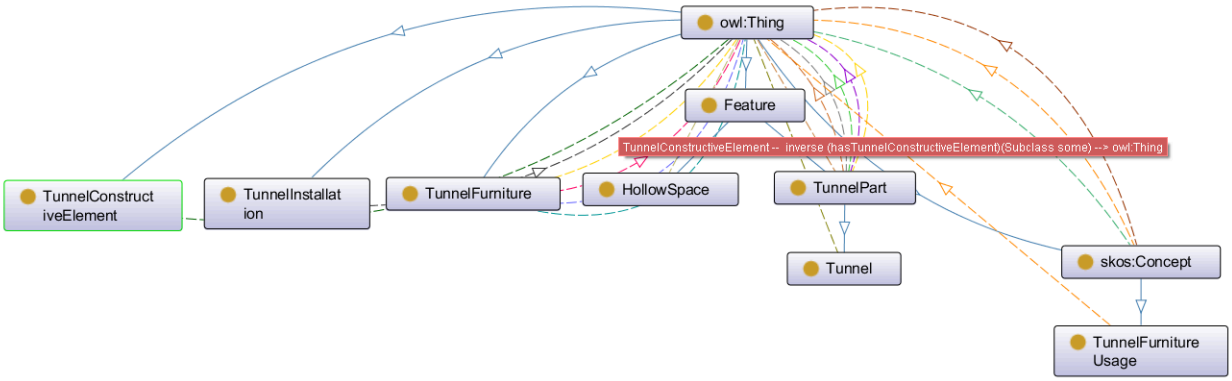


Figure 79 — Informal OWL drawing of CityGML's `Tunnel` class made by Protégé's *OntoGraph* plugin, layout by hand. Also shown are details of a selected relationship.

9.5.5. UML OWL diagrams

Most formal OWL specifications use UML or UML-like diagrams. For example the W3C Asset Description Metadata Schema (ADMS) (W3C TR vocab-adms) ontology's domain model is as per Figure 80 that shows what it terms a "UML model of ADMS classes and properties":

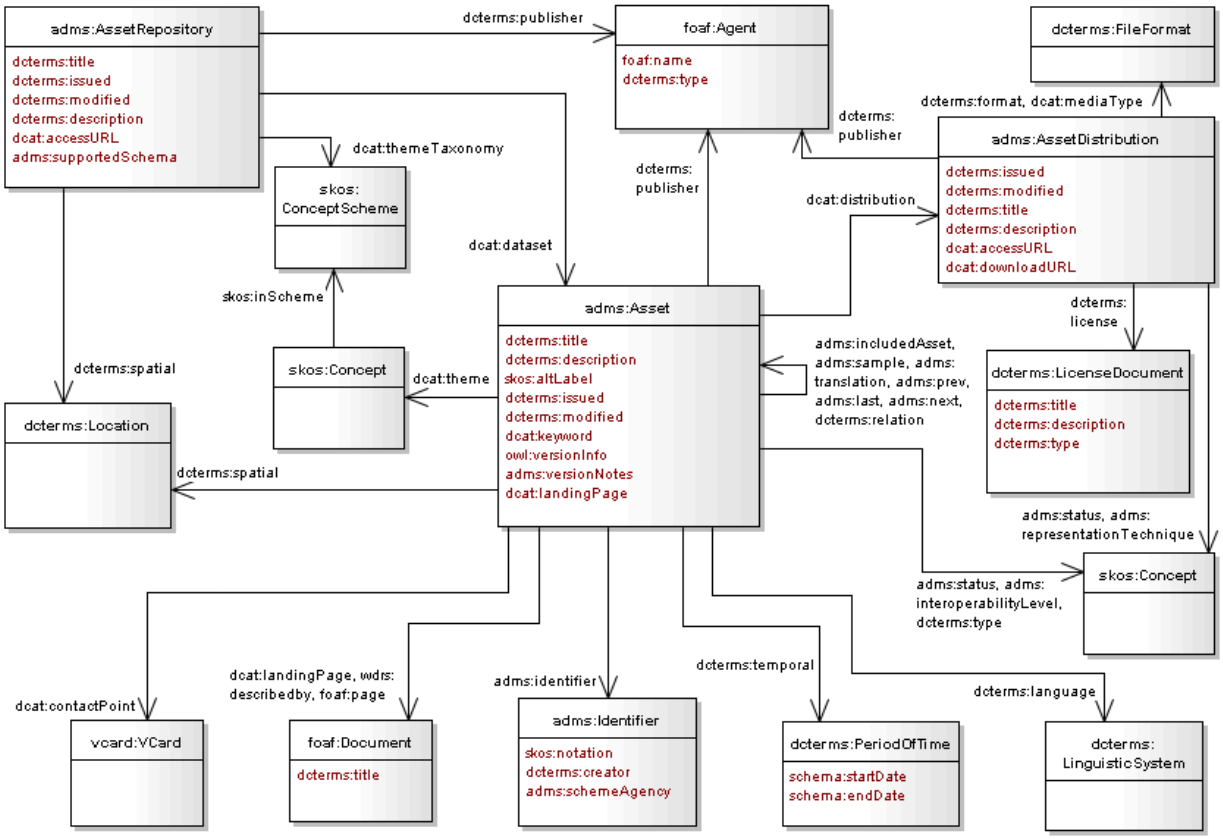


Figure 80 — Asset Description Metadata Schema (ADMS) ontology's domain mode as a "UML model of ADMS classes and properties". W3C TR vocab-adms, Clause 5

It seems, based on the styling, that the UML diagram in the figure above was drawn using Sparx Systems Enterprise Architect, a UML diagramming tool well known to OGC modellers.

Other examples of UML OWL diagrams are given in the following W3C Specifications:

- Dataset Catalogue Vocabulary, Version 2 (DCAT2)
  - Modern Sparx EA-style UML diagram
  - https://www.w3.org/TR/vocab-dcat-2/=UML_DCAT_All_Attr
- DCAT Version 1
  - Less formal UML diagram not drawn with Sparx's EA
  - https://www.w3.org/TR/2014/REC-vocab-dcat-20140116/=overview

9.5.6. UML OWL diagram tooling

In addition to drawing UML diagrams for OWL ontologies in tools such as Sparx' EA, some OWL modelling tools can auto-generate UML or UML-like diagrams. For example, TopQuadrant

TopBraid Composer can draw diagrams like the following, given the CityGML code in the listing above:

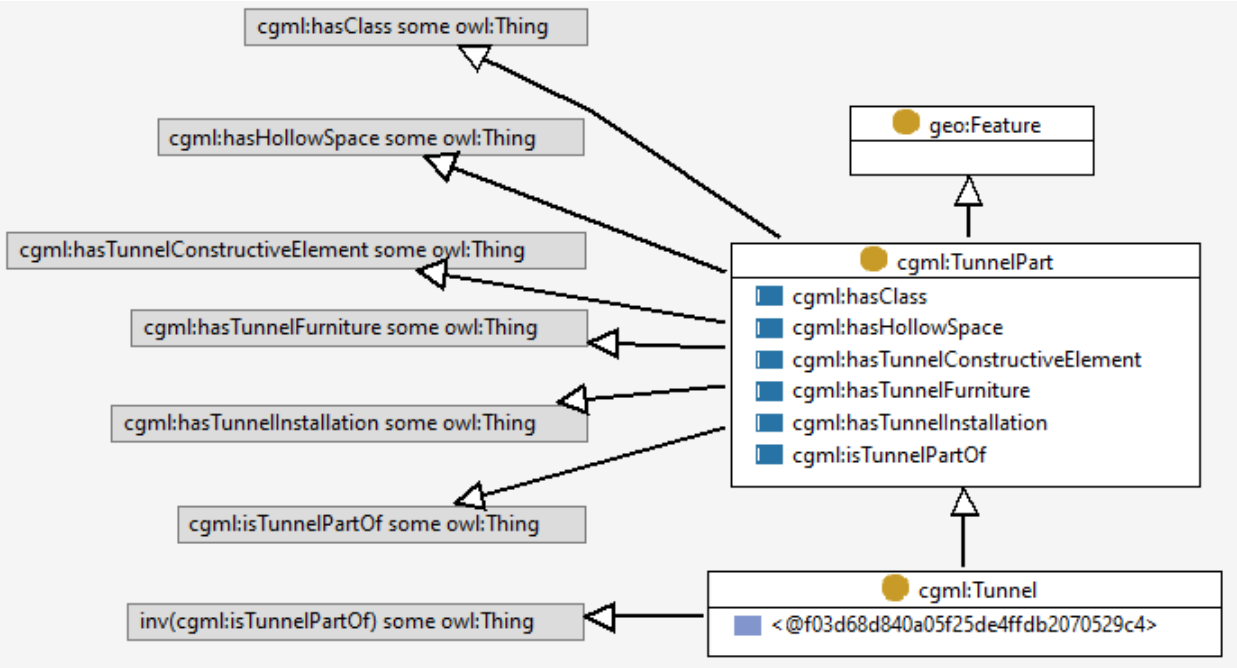


Figure 81 — TopBraid Composer's auto-drawn UML diagram of CityGML's `Tunnel` class ontology data

TopBraid Composer presents many options to the user for rearranging the diagram, hiding or showing elements and so on. What is shown above is a very simple rendering of only part of the CityGML `Tunnel` data, but more could be shown.

9.5.7. Native OWL diagrams

In addition to informal and UML diagrams, there are other kinds of formal OWL diagrams which are referred to here as 'native' OWL diagrams. These are diagrams that follow published specifications specific to OWL. There are several but the two demonstrated here are:

1. Lohmann et al. 2016 (Visual OWL)
2. GRAPHOL

9.5.8. Native OWL diagram: VOWL

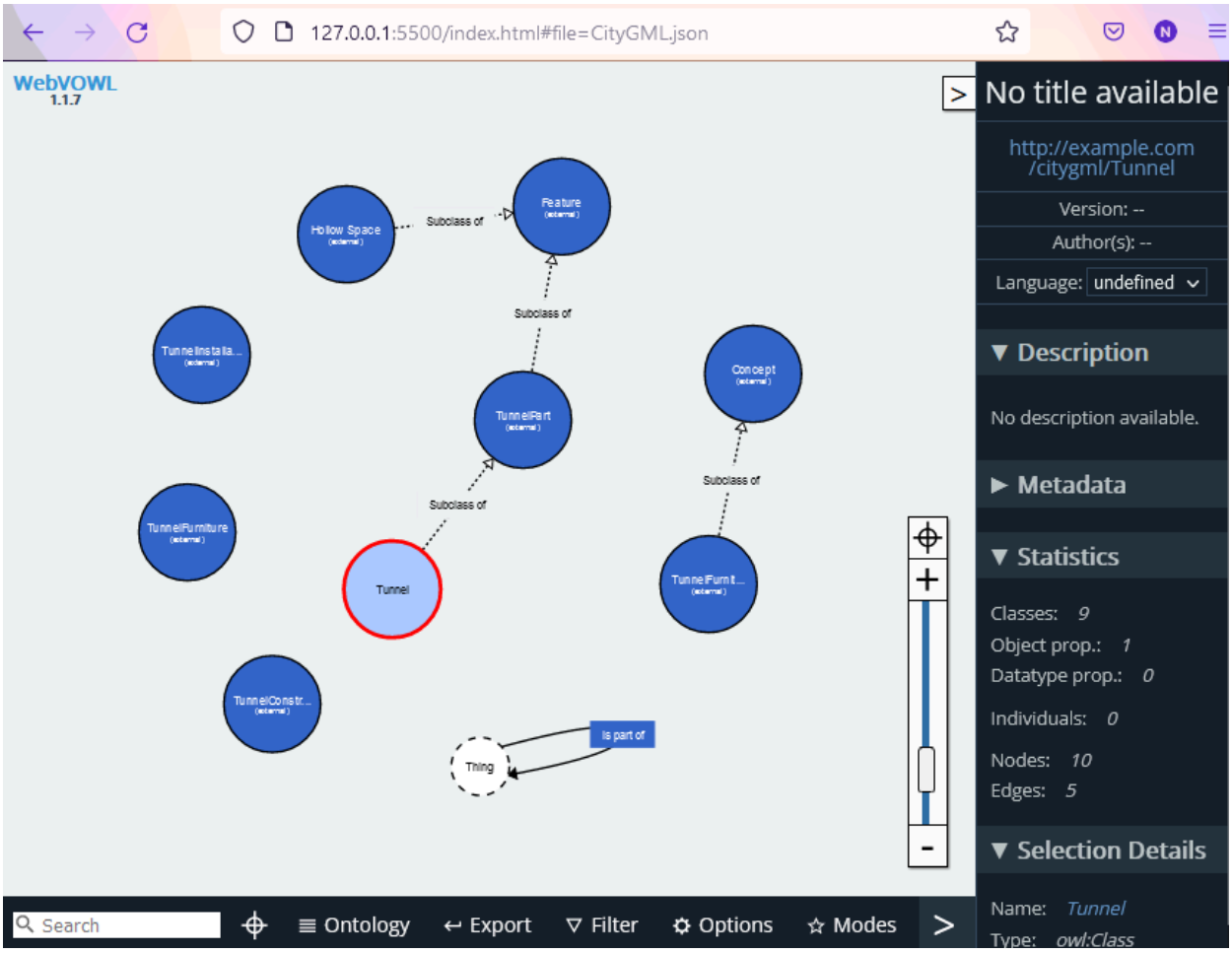


Figure 82 — A Visual OWL (VOWL) representation of the CityGML's `Tunnel` class ontology data. Auto-generated by WebVOWL

VOWL diagrams are commonly seen in ontology documentation resources, for example the *I-ADOPT Framework ontology*[13]. VOWL documentation says that it "focuses on the visualization of the ontology schema (i.e. the classes, properties and datatypes, sometimes called TBox), while it also includes recommendations on how to depict individuals and data values (the ABox)".

In the quick visualization of data above, a particularly useful diagram was not produced, however it seems clear that VOWL can visualize all/most OWL constructs so more fiddling with the tooling would likely produce reasonable results.

The VOWL developers used to provide an online free-to-use version of their tool but that is no-longer available, however desktop use is easy.

[13] https://w3id.org/iadopt/ont/0.9.0

9.5.9. Native OWL diagram: GRAPHOL

GRAPHOL is an OWL diagramming language which “builds on the Entity-Relationship model, but has a formal semantics and higher expressiveness. Notably, OWL 2 can be completely encoded in GRAPHOL”. The following GRAPHOL diagram was created based on the CityGML’s `Tunnel` class ontology data but drawn by hand using the Eddy GRAPHOL tool[14] supplied by a company providing commercial GRAPHOL support.

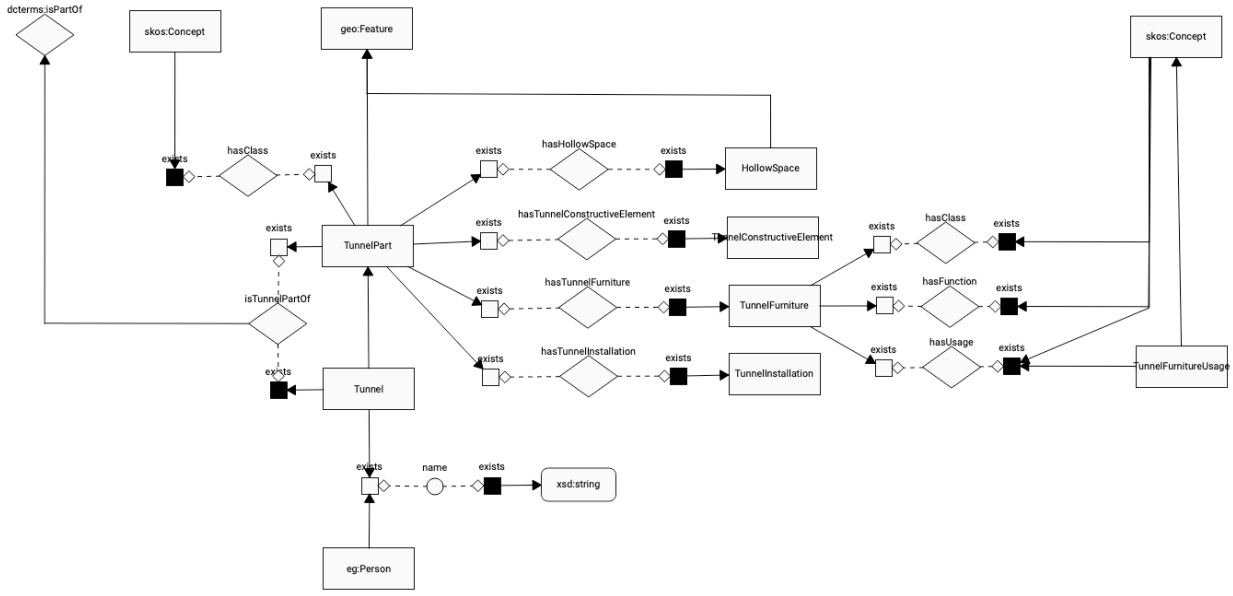


Figure 83 — A GRAPHOL representation of the CityGML’s `Tunnel` class ontology data. Hand drawn using the Eddy tool

Note that Eddy can turn GRAPHOL diagrams into standard OWL ontology documents but not the other way around as layout information is lost. This prevents the loading of pre-existing ontologies into Eddy and diagram rendering.

The above GRAPHOL diagram is “OWL complete” in that it claims to represent all elements of the OWL ontology it is diagramming with figure elements. Simplified versions of GRAPHOL diagrams can be auto-produced by supplementary GRAPHOL tooling. Such a ‘GRAPHOL Lite’ diagram is given below:

[14] CityGML’s `Tunnel` class ontology data

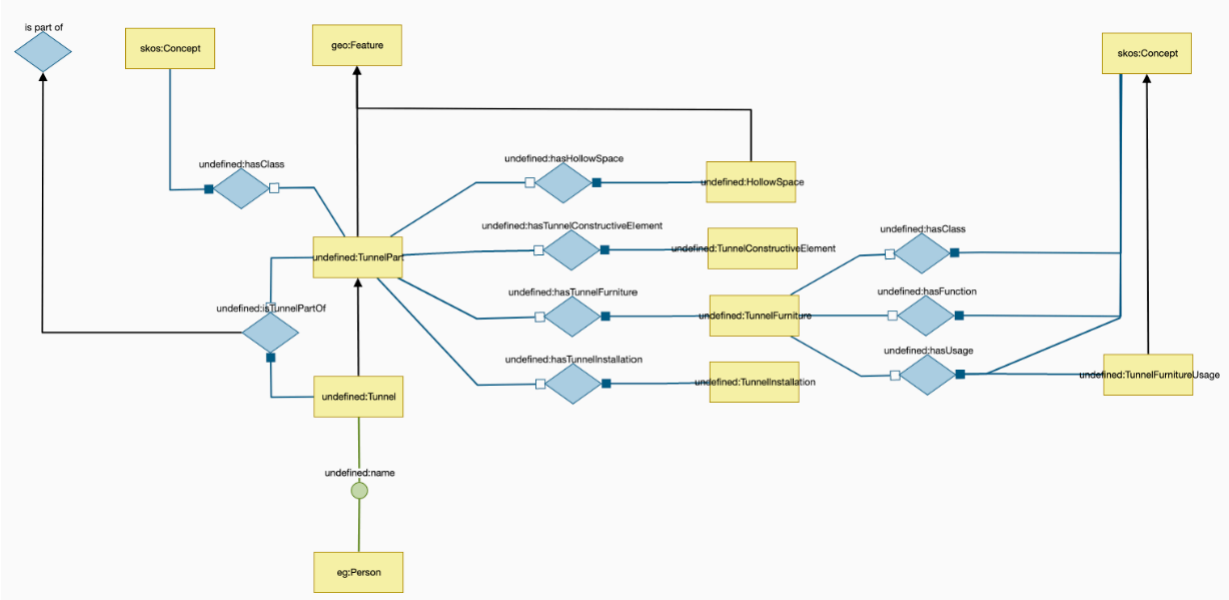


Figure 84 — A GRAPHOL Lite representation of the CityGML's `Tunnel` class ontology data auto-produced from the original GRAPHOL diagram

9.5.10. Diagram style summary

There are quite a number of ways to visualize OWL ontologies ranging from hand-drawn UML diagrams to auto-generated, OWL-specialized, graph diagrams.

Round-tripping (diagram → code → diagram or *vice versa*) is possible in some systems, however layouts are lost in the tools tested. It is suspected that there are diagramming round-tripping capabilities in some tools, such as *TopBraid Composer* but these capabilities weren't followed up in Testbed 17.

There seems to be quite an appetite for OWL diagramming, given the range of options and the fact that there are formal specifications and whole companies, such as the authors of *Eddy*, dedicated to it. If the OGC were interested in pursuing visual OWL modelling, more investigations here would probably be able to address most queries.

9.5.11. Diagrams within Standards' *Specification* documents

In work for the Australian/New Zealand 3D Cadastre Standard, per-element diagrams were placed into the *Specification* document that was generated from the Standard's ontology data. The diagrams were generated by hand and using a range of diagramming tools, such as Sparx Systems Enterprise Architect and the TopQuadrant TopBraid Composer EDG system's class visualizer.

These diagrams were placed *per element* within the *Specification*. The following three diagrams show informal, UML *Class* and UML *Package* diagrams.

7.3.1. Class: Cadastral Parcel

A single or multi area, or solid, above, or below the surface of the earth as specified through legislated process. A cadastral parcel can be described by a surface or solid and topological relationships with other parcels.

NOTE	Should a surface area only be mandatory for a primary cadastral parcel?

7.3.1.1. Subclass of

- 3D Spatial Unit
- No Title

7.3.1.2. Images

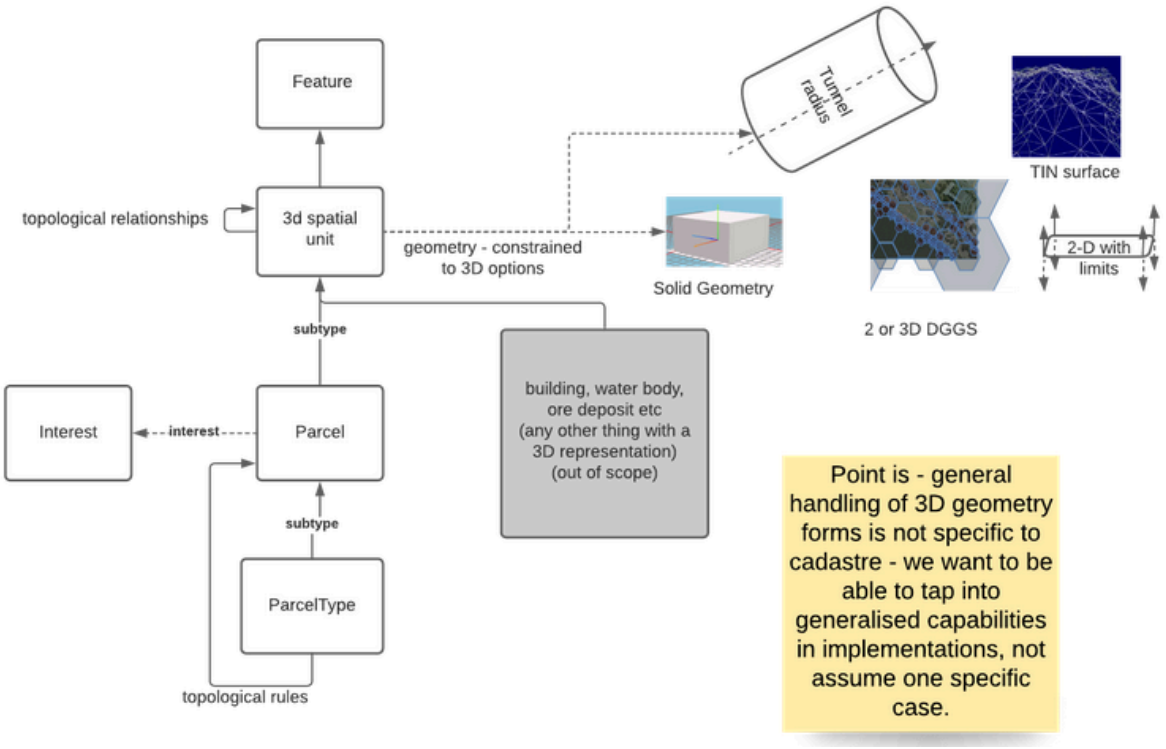


Figure 85 — An informal Class diagram within the draft Australian/New Zealand 3D Cadastre Standard, drawn by hand

7.6.15. Class: Survey Mark

A SurveyMark is a physical object which by its form defines a point on the surface of the Earth and which is stable during surveying operations.

NOTE	Specific functions may be subclasses or attributes depending on both consistency of terminology and need to describe specific constraints per function. Functional subclasses may not be disjoint.

7.6.15.1. Subclass of

- Survey Point

7.6.15.2. Images

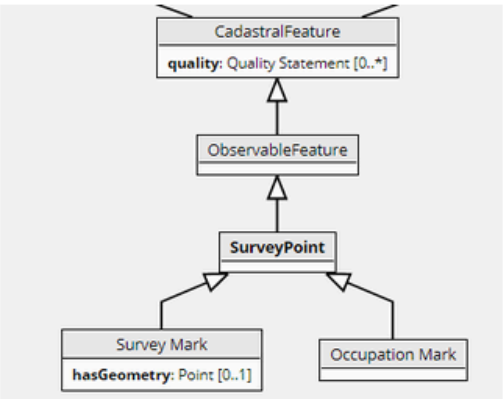


Figure 86 — An informal UML class hierarchy diagram, drawn using TopQuadrant's *EDG* system

7.5. Conformance Class: Provenance Profile

urn:x-evn-master:provenance_profile

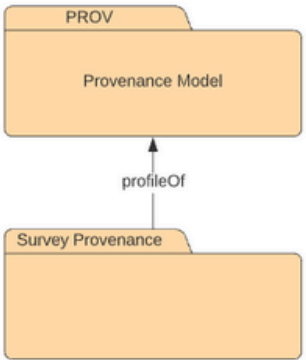


Figure 5. Package Dependencies Diagram

Format	image/png

Figure 87 — A UML Package diagram for `Conformance Class` objects in the draft Australian/New Zealand 3D Cadastre Standard, drawn by hand

These diagrams were associated with the Domain Model elements they represented by use of the *Exemplification Ontology* that provides "An ontology for the description of examples". The example descriptions for the above three diagrams were able to define:

- diagram metadata — title, description etc.
  - to be used for caption generation
- diagram resource technical descriptions
  - the format, file size & dimensions of images
  - used in auto-documentation generation
- example *role*
  - the role of the diagram v. the exemplified element
  - Roles of *canonical diagram* and *implementation diagram* are used in the ANZ 3D Cad model
- diagram Standard conformance
  - describes the conformance, if any, of the diagramming style to a diagramming specification, for example UML Class Diagram
  - using `dcterms:confromsTo`, conformance to ISO specifications such as ISO 19109 cd cad *ApplicationSchema* (ISO 19109:2015) are indicated
  - specialized profiles of diagramming are currently being created by the OGC Naming Authority and diagrams could indicate conformance to them. They indicate a diagramming formalism and purpose and are the *Class Context* and *Package Dependencies* diagramming profiles.

This use of the *Exemplification Ontology*'s packaging of diagrams within a Standard's Domain Model is the same as its use for packaging code and other text data examples.

9.6. Conclusion

This Testbed activity purposely explored the limits of what is currently possible regarding Model-Driven Standards and what may soon be possible with enhanced conceptual modelling of Standards & Specifications, updated tooling and new examples of OGC & ISO Standards to test modelling for.

It is clear that:

- various OWL models can be used to model *all* of the information within a Standard
    - including all parts of a Specification
    - and Standards' parts' relations
- OWL models can be shown graphically in a number of ways, some which look similar to, or are, UML
- tooling is now available that can produce Standards documents from OWL models
    - next-generations of this tooling may be generic enough for many standards

What is not yet known is to what extent current standards editors find it possible to adopt OWL-based approaches to standards content modelling, both the Domain Models and other elements.

It is hoped that with the testing of oMDS approaches for GeoSPARQL 1.2 and perhaps other standards, some OGC, and perhaps ISO TC 211, members will gain first-hand experience with developing ontology Model-Driven Standards and will be able to relate their pros and cons to their member organizations shortly, perhaps within 2022.

10

TESTBED SUB-TASK D144

10 TESTBED SUB-TASK D144

10.1. General

The D144 sub-task was run by Ribose Limited to investigate model-based approaches to publication of standards and content.

The participants adopted a workflow based on Metanorma and ShapeChange to prototype two recent OGC Standards that were accompanied by formal UML-based conceptual models, employed as use-cases in this task.

- OGC 20-010 *CityGML 3.0 Part 1: Conceptual Model Standard* (Draft: 2021-03-02; Final: 2021-09-13)
- OGC 20-040r3 and ISO 19170-1:2021 *Geographic information — Discrete Global Grid Systems Specifications — Part 1: Core Reference System and Operations, and Equal Area Earth Reference System* (dual published, 2021-05-11)

These two standards are considered appropriate for this prototype for the following reasons:

1. Both are model-based standards, each based on a single set of consistent UML conceptual models.
2. The UML conceptual models are made available in a public, open-source manner. In CityGML, the conceptual model is made available by the CityGML SWG; in DGGS, by ISO/TC 211.
3. Both published standards heavily rely on model annotations documented within the conceptual model. In both cases the class diagrams had been derived from the conceptual models.
4. The conceptual models of these standards are both documented in a Sparx Systems Enterprise Architect Project (EAP) file that could be used as the basis for application of the MDA-based UGAS methodology.

The conceptual models for both CityGML 3.0 and DGGS were evaluated as input to a PIM-to-PSM transformation process, involving ShapeChange for PSM generation and Metanorma for model-driven standards document generation, with the objective of replicating the structure, content, and style of the existing standards document.

The ready availability of pre-final (CityGML) and final (DGGS) text enabled the analysis of two different target forms for formal presentation of conceptual model content as traditional text-based documents containing embedded UML class diagrams.

Table 7 — Software used for D144 prototypes

PROTOTYPE	PIM (CONCEPTUAL MODEL)	PSM	SOFTWARE
Prototype A1	CityGML 3.0	(OGC Standard)	Metanorma
Prototype A2	DGGS	(OGC Standard / ISO Standard)	Metanorma
Prototype B1	CityGML 3.0	GML / XML Schema	ShapeChange
Prototype B2	CityGML 3.0	JSON Schema	ShapeChange
Prototype B3	CityGML 3.0	RDF	ShapeChange

The OGC CityGML Standards Working Group (SWG) had employed a former release of ShapeChange to generate XSD-based schema files as part of a development activity to simultaneously release an XSD-based encoding of the CityGML 3.0 Conceptual Model. Specific software modifications had been made to a branch of ShapeChange in that effort. An early objective of this task was to evaluate those modifications, integrate them into the current ShapeChange baseline, and use the result as the single ShapeChange software baseline for all PIM-to-PSM evaluations.

10.2. CityGML 3.0

10.2.1. General

OGC 20-010, the CityGML 3.0 Part 1 Conceptual Model Standard, defines a common semantic information model for the representation of 3D urban objects that can be shared over different applications. CityGML defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantic, and appearance properties. “City” is broadly defined to comprise not just built structures, but also elevation, vegetation, water bodies, city furniture, and more. Included are generalization hierarchies between thematic classes, aggregations, relations between objects, and spatial properties.

The CityGML conceptual model builds on a number of conceptual models taken from other standards (Figure 88):

- ISO 19103:2015
- ISO 19107:2019
- ISO 19108:2002

- ISO 19109:2015
- ISO 19111:2019
- ISO 19123:2005
- OASIS xAL v3.0 (OASIS CIQ v3.0)

The OGC CityGML SWG is completing work to formalize a corresponding CityGML 3.0 Implementation Specification for GML. The creation of further implementation specifications, particularly for JSON — accompanied by potential harmonization with the OGC CityJSON Community Standard (based on CityGML 2.1) — is part of the SWG work program.

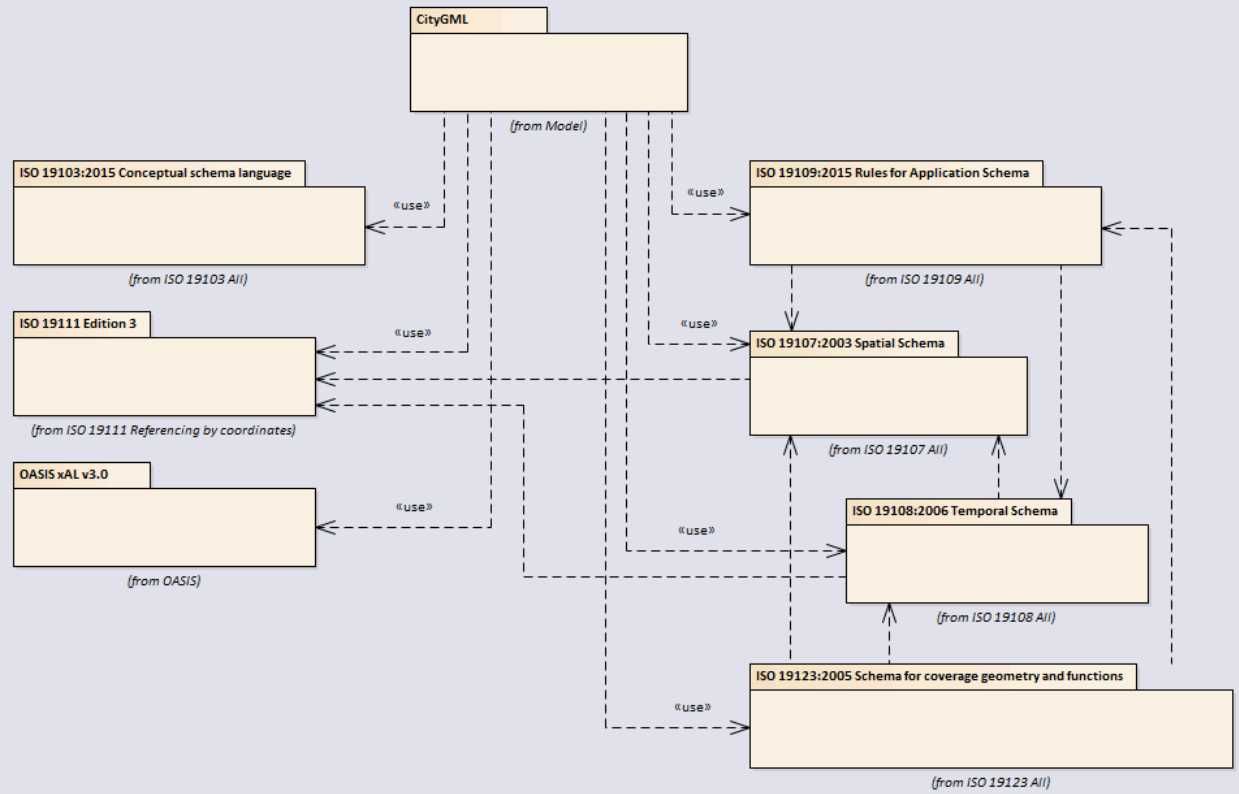


Figure 88 — CityGML 3.0 Part 1 — Conceptual Model UML Package Dependencies

10.2.2. Goals and challenges

There are two stated goals in D144 in respect to CityGML 3.0:

1. The CityGML 3.0 conceptual model is a data representation model that is meant to be a PIM. It is an abstract model not meant to be (and cannot be) directly used or implemented by machines.

    a) In order to make it useable, we need to transform it into PSMs: machine-useable encodings, such as XML schemas, JSON schemas and RDF so that machines can utilize this conceptual model.

2. The CityGML 3.0 conceptual model needs to be published as an OGC Standard, which include model annotations, text and ModSpec models.

    a) The standard is composed of the following components:

        i) OGC standard metadata, introductory and guidance text are encoded in Metanorma AsciiDoc formats.

        ii) The UML portion of the conceptual model was developed in UML using Sparx Systems Enterprise Architect and exists as an EAP file (a proprietary format)

        iii) Some model annotations are stored within the UML portion, and other portions exist as Metanorma AsciiDoc text.

    b) The challenge is to create a single-step workflow that creates the CityGML 3.0 conceptual model standard.

Prototype A1 (Clause 10.4), B1 (Clause 10.6), B2 (Clause 10.7) and B3 (Clause 10.8) were implemented to achieve these goals.

10.3. Discrete Global Grid Systems (DGGS)

10.3.1. General

ISO 19170-1:2021 defines common classes for spatiotemporal geometry, topology, and reference systems using identifiers. The DGGS Core Reference system is based on zonal identifiers for structured geometries that may be spatiotemporal. ISO 19170-1 supports the specification of standardized DGGS infrastructures that enable the integrated analysis of very large, multi-source, multi-resolution, multi-dimensional, distributed geospatial data. Interoperability between DGGS-based implementation standards is anticipated in the future (at the present time only XML Schema is supported), as well as extension interface encodings of OGC Web Services.

The DGGS Conceptual Schema is published as part of single Enterprise Architect (EA) project (EAP) file maintained by the ISO/TC 211 Harmonized Model Maintenance Group (HMMG: https://github.com/ISO-TC211/HMMG).

The DGGS conceptual model is a data representation model that is meant to be a PIM.

The DGGS conceptual model builds on a number of conceptual models taken from other standards (Figure 89):

- ISO 19107:2019
- ISO 19111:2019
- ISO 19112:2019
- ISO 19115-1:2014
- ISO 19156:2011

The UML package “ISO 19170-1 Edition 1”, along with those appropriate UML packages upon which it has dependencies, was extracted for use in this task.

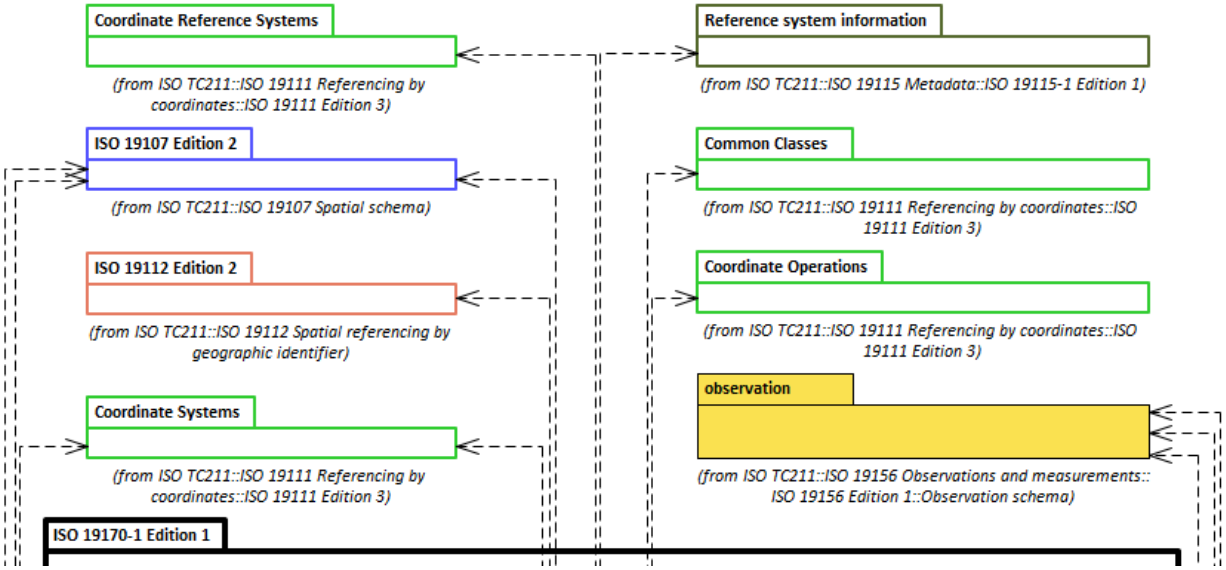


Figure 89 — ISO 19170-1:2021 UML Package Dependencies

10.3.2. Goals and challenges

There are two stated goals in D144 in respect to DGGS:

1. The DGGS conceptual model needs to have its contents and guidance publishable as an OGC document together with ModSpec requirements
2. The DGGS conceptual model needs to have its contents and guidance publishable as an ISO document together with ModSpec requirements

Prototype A2 (Clause 10.5) was implemented to achieve these goals.

10.4. Prototype A1: CityGML 3.0 PIM to Standard derivation

10.4.1. General

The CityGML 3.0 Conceptual Model is published in the form of a set of XMI files that were generated from a single EAP file, and the source EAP file itself.

This set of files includes:

1. a single XMI file version with all CityGML 3.0 packages integrated but no references;
2. a per-package collection of XMI files for each package in the CityGML 3.0 UML model;
3. XMI files for the ISO/TC 211 and OASIS references; and,
4. a single (large) XMI file with CityGML 3.0 UML model plus the ISO/TC 211 and OASIS references.

Models upon which there are UML package dependencies are included in 3) and 4).

The CityGML 3.0 CM incorporates the full ISO/TC 211 Harmonized Model (ISO/TC 211 HM), published and managed by the ISO/TC 211 Harmonized Model Maintenance Group (HMMG). The ISO/TC 211 HM, in some cases, contains multiple editions of models.

In this Testbed activity, the published CityGML 3.0 EAP file was evaluated for its readiness as a PIM conceptual model for use in the derivation of PSM implementation schemas, and PSM implementation standards. Multiple issues were identified, in some cases resulting in changes to the original model, as detailed in Clause 8.2.

The revised CityGML 3.0 Conceptual Model was used as the source information for all CityGML 3.0 tasks.

The EAP file used for these prototypes is available at:

- EAP format

The prototypes A1 and B1, B2, B3 directly utilized XMI files exported from this EAP file as input to generate MDSs as outlined in Figure 90.

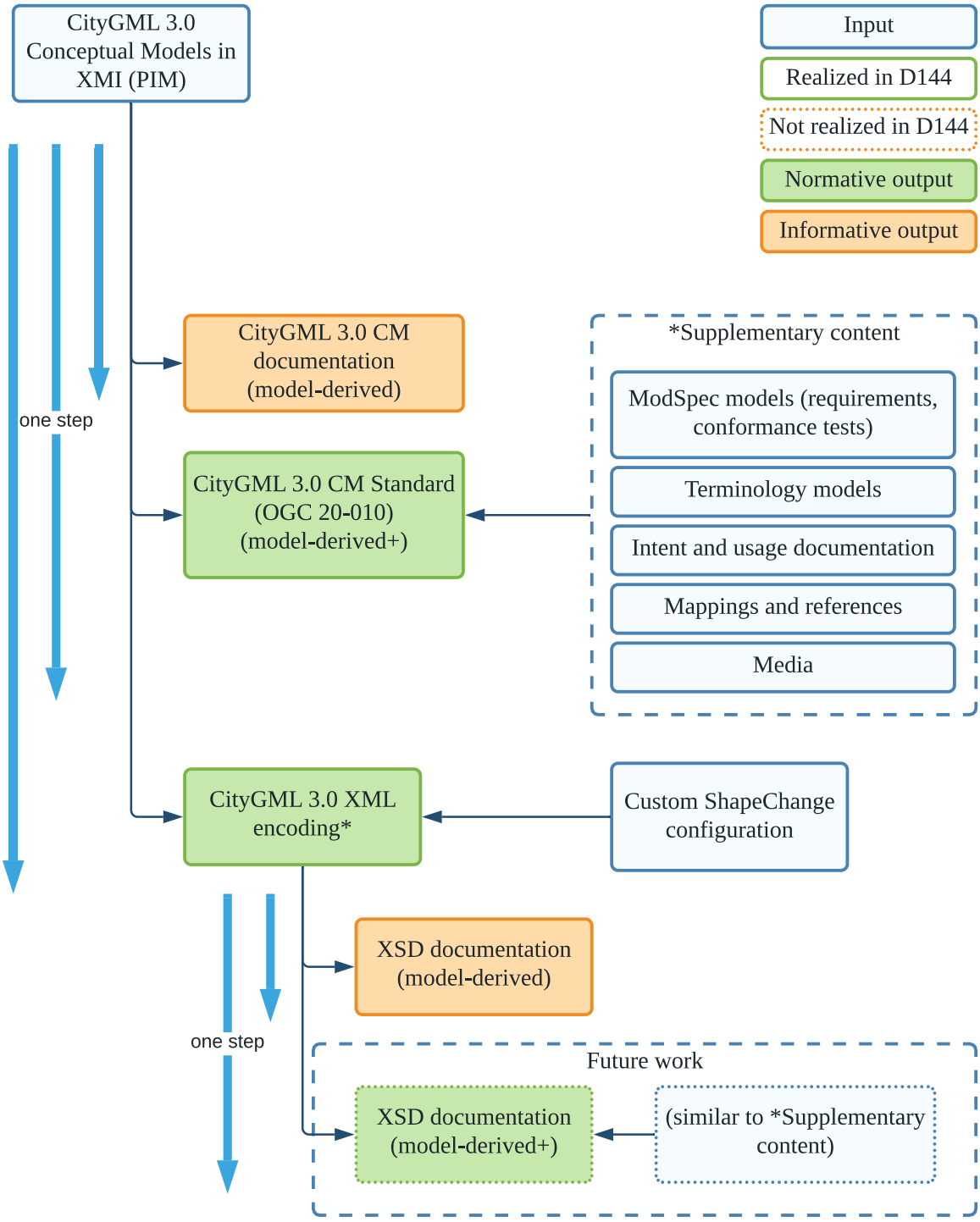


Figure 90 — Standards and deliverables derived from the CityGML 3.0 conceptual models (PIM to PSM)

The prototype is implemented and open-sourced at: https://github.com/metanorma/ogc-citygml-xmi (location change pending).

10.4.2. Methodology

The overall process for generating the OGC CityGML 3.0 Conceptual Model standard (20-010) is captured in Figure 91.

This is a single generation step that generates an MDS from a PIM XMI and corresponding text.

During development of the prototype, this generation step was enabled by iterative development and debugging, as issues in the encoding of the source information model were identified, and capabilities in the MDA software were enhanced.

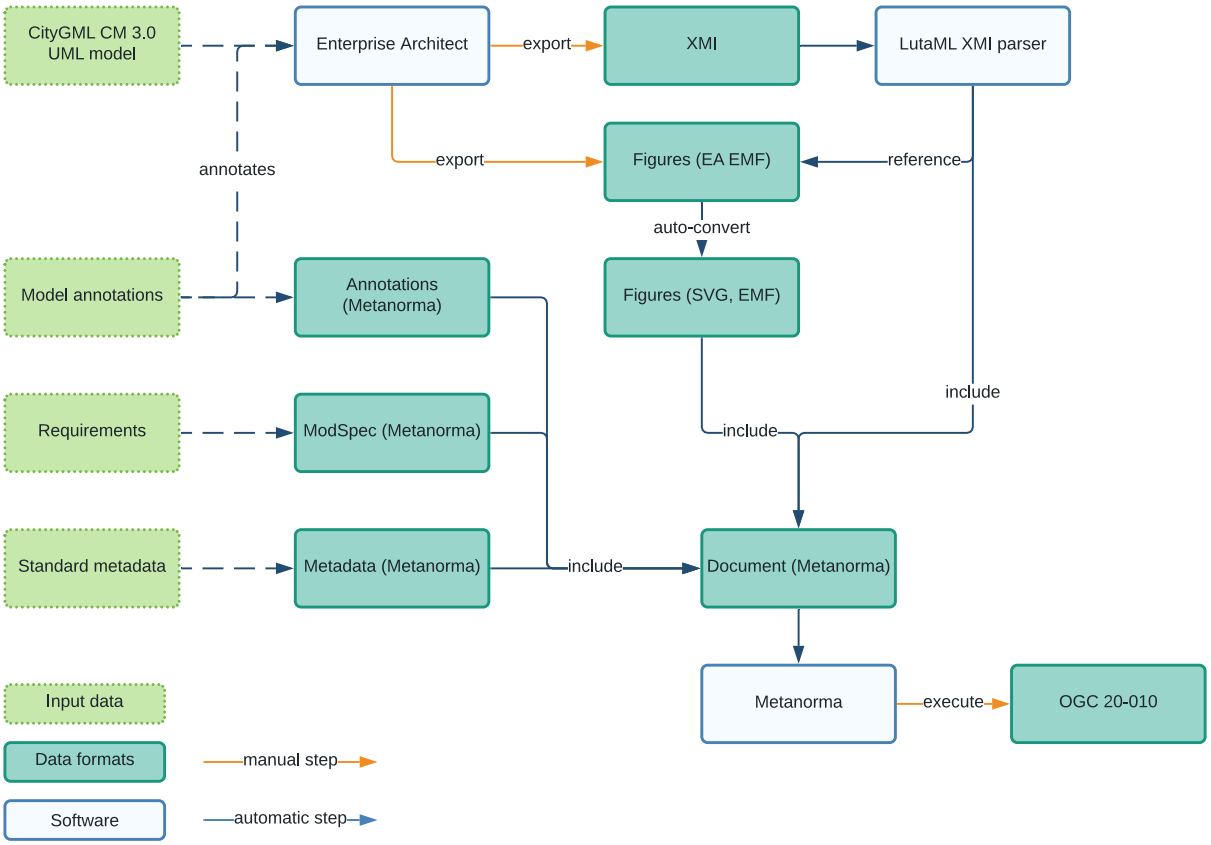


Figure 91 — CityGML 3.0 MDS information model flow

As discussed in Clause 4.14, any MDS involves merging derived and supplementary truth, and authoring the supplementary truth is an iterative process, dependent on the derived truth.

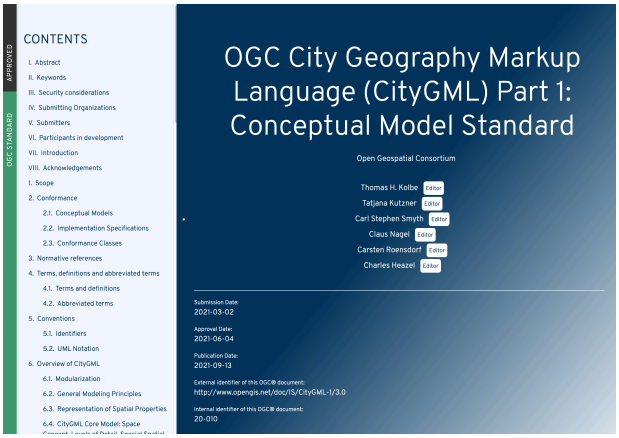
Generating an MDS therefore involves the following steps:

- Export: Making the PIM available for processing (Clause 10.4.3)
- Authoring: Making the supplementary truth available for processing (Clause 10.4.4)
- Data parsing: Parsing the truth of the model into derived truth in the document (Clause 10.4.5)

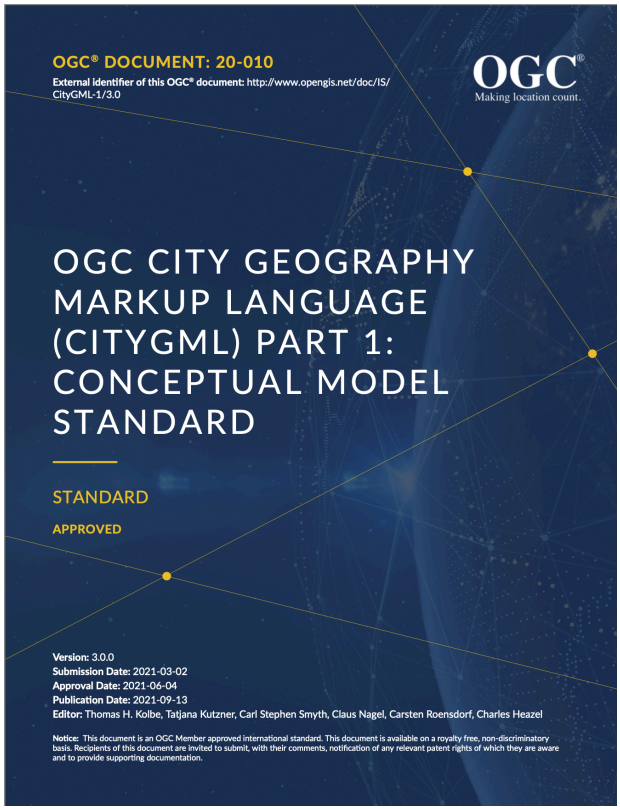
- Integrating: Merging derived and supplementary truth into the target document (Clause 10.4.6)
- Rendering: Generating human-consumable presentations of the target document (Clause 10.4.7)

The approach shown in Figure 90 has proven successful culminating in the OGC CityGML 3.0 Conceptual Model standard (20-010) being officially published by OGC on 2021-09-13 utilizing the toolchain developed in sub-task D144 (Table 8).

Table 8 — OGC 20-010: CityGML 3.0 Conceptual Model standard in HTML and PDF formats



OGC 20-010: CityGML 3.0 Conceptual Model standard in HTML format



OGC 20-010: CityGML 3.0 Conceptual Model standard in PDF format

10.4.3. Export

Making source data available for MDA involves exporting the information models in a standardized interoperable format from the model authoring tool.

The main truth for the CityGML 3.0 conceptual model (noted in Clause 10.4.1) is the set of XMI files describing the CityGML UML model. The initial step in processing is to export those XMI files (see Figure 92).

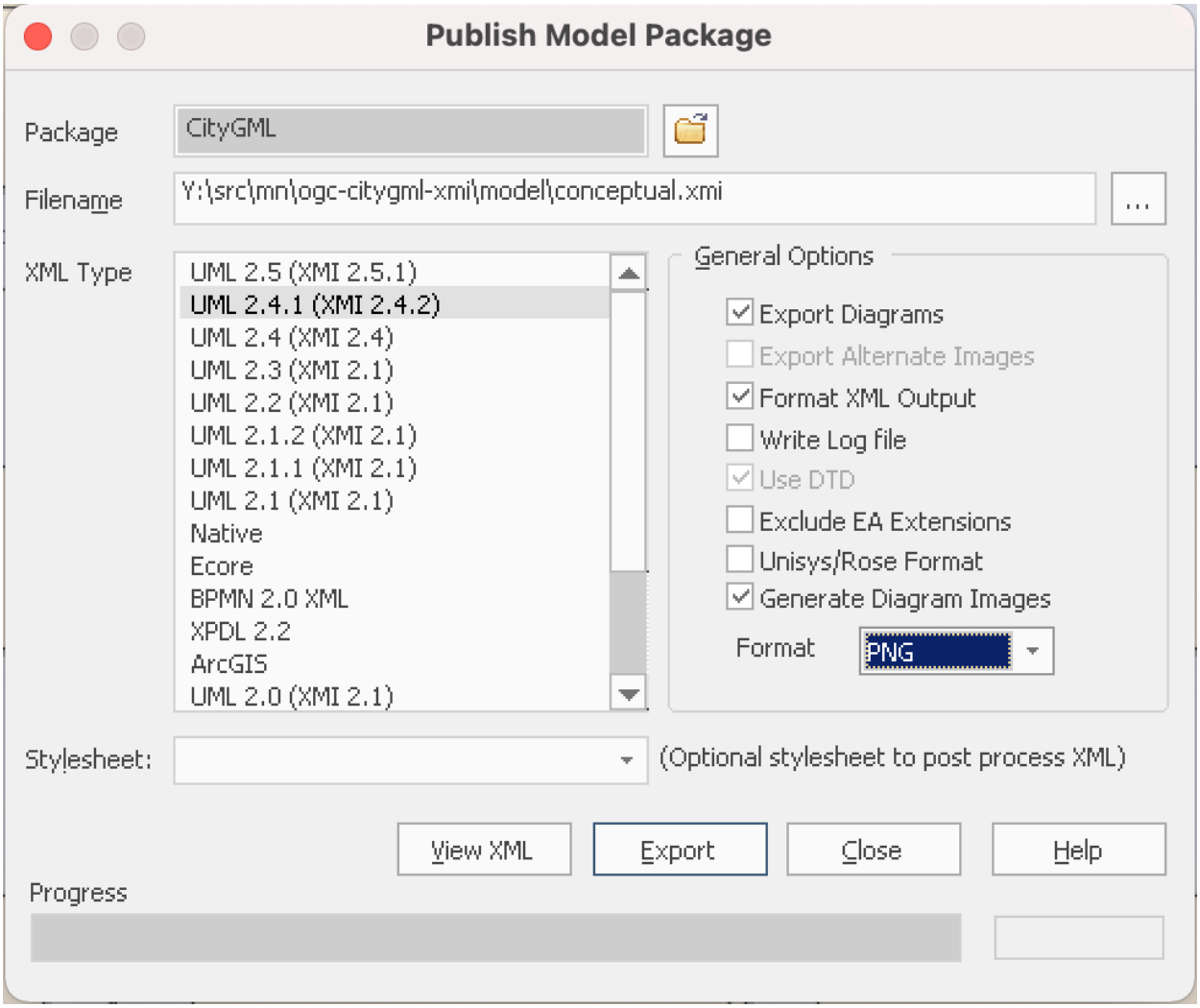


Figure 92 — Export options for CityGML 3.0 CM EA model to XMI for Metanorma processing

A secondary set of source data are the UML diagrams located inside the same Sparx Systems Enterprise Architect package, which provide visual representations of UML classes described in the XMI files.

NOTE: Information given in the UML diagrams are separate truths from the model definitions — diagrams do not represent the models themselves.

The UML diagrams are exported by Sparx Systems Enterprise Architect at the same XMI export step into a separate directory. These diagrams are named according to proprietary GUIDs of the diagram objects (e.g., `EAID_0FCA1673_FDB0_45de_875B_F1733FD7FA88.emf`), which makes them incredibly difficult to use outside the program.

Typically, UML diagrams are best rendered in vector formats since they contain heavy use of shapes. As the diagrams are stored inside an EAP package, the export functionality of those diagrams depend on Sparx Systems Enterprise Architect.

However, this caused the following problems in the attempt to create an MDS that is accessible cross-platform and across multiple formats:

1. Sparx Systems Enterprise Architect can only export in the outdated, obsolete and proprietary Microsoft EMF format, which cannot be used in applications outside Microsoft Windows, as described in Clause 10.4.3. Worse, the EA implementation of EMF is not even compliant to the official Microsoft EMF specification — a correct implementation of EMF will display some objects in the EMF in a vertically flipped manner.

2. Sparx Systems Enterprise Architect allows exporting images to raster graphics format like PNG, but the resolution is lower than desired on modern displays, and that many standards organizations require vector formats for the possibility to translate text.

To overcome the hurdles above, the prototype implemented the following:

1. The Sparx Systems Enterprise Architect generated EMF images were “normalized” via a modified version of the `libemg2svg` EMF library, so that they are conformant to the latest Microsoft EMF specification.

2. The Sparx Systems Enterprise Architect generated EMF images are then converted into the W3C SVG vector format which is a standard supported by most browsers across all major platforms.

3. These SVGs are then imported into the different MDS output formats, such as PDF and HTML, and for Word output, these SVGs are converted back to a compliant EMF format (Microsoft Word does not support SVGs, only EMF).

    - EA diagrams were found in violation of the Microsoft EMF specification in several ways. These EMF files caused incompatibilities with non-Microsoft-Word applications and renderers, which resulted in heavy development effort in forking the `libemf2svg` library (https://github.com/metanorma/libemf2svg). The detailed fixes are documented in Annex G.

These steps enabled usage of a single source (the EAP file) as the authoritative source for UML diagrams in the automated generation process required in the MDS.

10.4.4. Authoring

Supplementary information in an MDS normally includes such material as bibliographies, terminological definitions, tutorial guidance, annexes, and prefatory material, which form part of a document presenting and explaining the model.

This information was written in Metanorma, using the OGC flavor of the Metanorma AsciiDoc markup language. Where the supplementary information references specific model artifacts (annotating them), cross-references from Metanorma to the model become necessary; those cross-references are part of the integration of derived and supplementary truth, and are discussed in Clause 10.4.6.

Two additional sources of supplementary information are also identified in Figure 91, and were likewise authored in Metanorma:

- Metadata about the document were authored in the Metanorma AsciiDoc document header as key-value pairs, and is used to generate a bibliographic description of the MDS, which can be machine-processed in library systems.
- Requirements conforming to OGC ModSpec were authored in Metanorma using Metanorma AsciiDoc or YAML (Annex E), as part of the current document. The ModSpec requirements covered the expected range of normative statements and validation of an OGC document, including “permissions” (Figure 93), “requirements” (Figure 94), “requirement classes” (Figure 95), and “conformance classes” and “conformance tests” (Figure 96, referencing the requirement in Figure 94).

PERMISSION 1	
/per/core/classes	
For each UML class defined or referenced in CityGML Conceptual Model:	
A	An Implementation Specification MAY represent that class as a null class with no attributes, associations, or definition.
B	An Implementation Specification MAY represent an association of the UML class with a null association.
C	An Implementation Specification MAY represent an attribute of the UML class with a null attribute.
D	Whenever a null element is used to represent a concept from the Conceptual Model, the Implementation Specification SHOULD document that mapping and provide an explanation for why that concept was not implemented.

Figure 93 — ModSpec Permission in CityGML

REQUIREMENT 1	
/req/core/classes	
For each UML class defined or referenced in the Core Package:	
A	The Implementation Specification SHALL contain an element which represents the same concept as that defined for the UML class.
B	The Implementation Specification SHALL represent associations with the same source, target, direction, roles, and multiplicities as those of the UML class.
C	The implementation Specification SHALL represent the attributes of the UML class including the name, definition, type, and multiplicity.
D	The implementation Specification SHALL represent the attributes of all superclasses of the UML class including the name, definition, type, and multiplicity.
E	The implementation Specification SHALL represent the associations of all superclasses of the UML class including the source, target, direction, roles, and multiplicity.
F	The Implementation Specification SHALL specify how an implementation observes all constraints the Conceptual Model imposes on the UML class.

Figure 94 — ModSpec Requirement in CityGML

REQUIREMENTS CLASS 1	
http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-core	
Target type	Implementation Specification
Dependency	ISO 19103:2015
Dependency	ISO 19107:2003
Dependency	ISO 19109:2015
Dependency	ISO 19111:2019
Dependency	ISO 19123:2005
Dependency	OASIS xAL v3.0

Figure 95 — ModSpec Requirements Class in CityGML

A.2. Conformance Class Core

Abstract test A.1	
/ats/core/classes	
Requirement	/req/core/classes
Test purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Test method type	Manual Inspection
Test method	a. For each UML class defined or referenced in the Tunnel Package: 1. Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class. 2. Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same source, target, direction, roles, and multiplicies as those documented in the Conceptual Model. 3. Validate that the data element has the same properties (attributes) as those specified for the UML class. Validate that those properties have the same name, definition, type, and multiplicity of those documented in the Conceptual Model. 4. Validate that the properties of the data element include those of all superclasses of the UML class as documented in the Conceptual Model. Validate that those properties have the same name, definition, type, and multiplicity of those documented in the Conceptual Model 5. Validate that the associations represented for the data element include those of all superclasses of the UML class as documented in the Conceptual Model. Validate that those representations have the same source, target, roles, and multiplicity of those documented in the Conceptual Model 6. Validate that the Implementation Specification enforces all constraints imposed on the UML class by the Conceptual Model

Top

Figure 96 — ModSpec Conformance Classes and Conformance Tests in CityGML

The use of a structured, metadata-rich representation of ModSpec in OGC Metadata, as shown in Annex E, allows for semantic detail and interdependency of requirements to be captured, and is considered as a model-driven approach to standards content.

As noted in Clause 4.17.4, the modular intent of ModSpec encourages ModSpec requirements to be maintained in a document-external tool, and reused between different documents. That means that in future iterations, ModSpec requirements could also be exported from a tool for consumption by Metanorma, and potentially parsed from a non-Metanorma format for tight integration — just as is done for the derived truth of the document.

10.4.5. Data parsing

Processing the XMI file is done under the Metanorma approach to MDA by LutaML (Clause 7.2), LutaML returns to Metanorma an array of objects, one for each of the objects in the source file parsed by LutaML, with a plugin structure to deal with the range of formats LutaML is called on to process (`lutaml-xmi`, in this instance).

Metanorma then uses Liquid directives to iterate through those objects, and insert information from them into Metanorma AsciiDoc templates. These templates are how information from the model is incorporated into the MDS as derived truth.

In the CityGML repository (https://github.com/metanorma/ogc-citygml-xmi) the source truth is stored in the directory `/xmi-full`, containing both the complete XMI file, and the exported UML diagrams under `/xmi-full/Images`.

Access to LutaML was done through the `lutaml_uml_datamodel_description` command (documented in Clause 7.2), which parses UML class information from a nominated XMI file, one UML package at a time.

Configuration files were used to specify which packages to render for each command call, in which sequence, and whether to display them at entity level (Clause 7, configuration file `lutaml_packages.yml`, Figure 97), or at data dictionary level (Clause 8, configuration file `lutaml_data_dictionary.yml`, Figure 98).

Custom configuration was proved necessary as CityGML 3.0 does not follow the LutaML default of iterating through all available packages in alphabetical and hierarchical order.

Instead, the order of classes in CityGML 3.0 has had to follow the standard document structure of CityGML 2.0, for backwards compatibility, with newly added classes documented out of sequence as add-ons to that structure.

The rendering style of these clauses that contain UML model information in CityGML 3.0 were incorporated into LutaML for the usage in future OGC MDS.

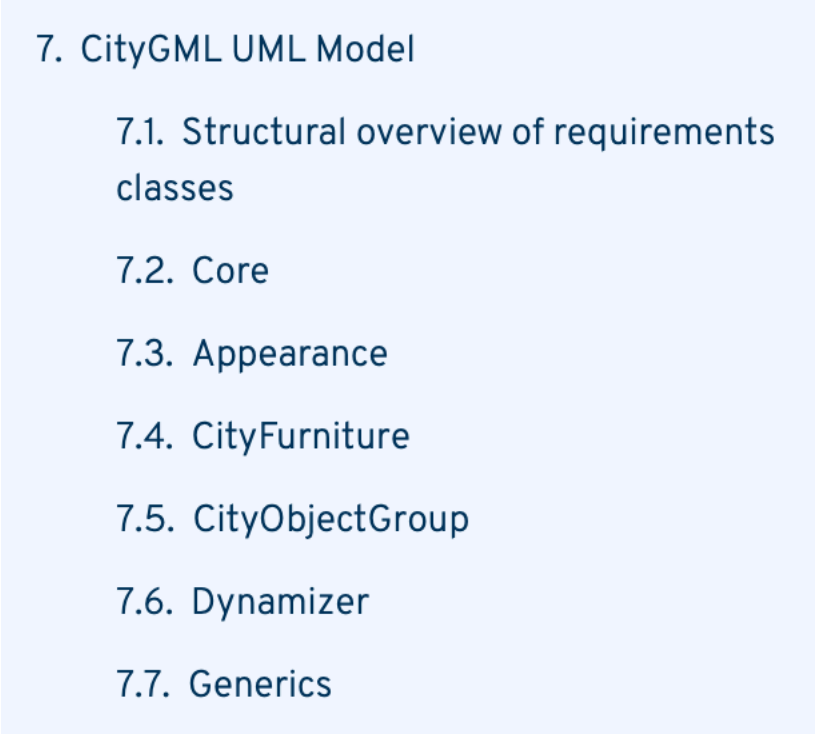


Figure 97 — Contents of entity level rendering of CityGML UML model

Figure 98 — Contents of data dictionary rendering of CityGML UML model

10.4.6. Integrating

A wide range of information is integrated into the target document, and this information is organized into separate directories in the CityGML repository (https://github.com/metanorma/ogc-citygml-xmi), under `sources/standard`:

- The main Metanorma AsciiDoc document (`20-010.adoc`), containing document metadata and directives to include sections contained in the document.
- The Metanorma AsciiDoc documents for each section in the standard (`sections/*.adoc`).
- A Metanorma AsciiDoc document capturing data dictionary information from ISO 19109:2015 (`data-dictionaries`), expressed as a set of Metanorma AsciiDoc tables.

  Unlike derived truth, this information is not derived from a model automatically, since the classes involved were not included in the source model. This information has been authored manually.
- The allowed thematic surface boundaries for various classes (`boundaries`), added to the presentation of the CityGML data dictionary as supplementary data, since that information was not directly included in the source model: again, the Metanorma AsciiDoc has been authored manually.
- Generic ModSpec requirements (not specific to the information models), each expressed as a separate file of Metanorma AsciiDoc, are included into the section documents at the appropriate point (`abstract_test`, `recommendations`, `requirements`).
- Non-model-generated images (`images`) and figures (`figures`) are included in the document as supplementary truths, as distinct from the UML diagrams exported into `/xmi-full/Images` as derived truths.

Supplementary truth is incorporated into the target document through standard AsciiDoc commands:

- `image::` for images and figures
- `include::` for content.

Supplementary truth that is interleaved with derived truth is interpolated into the LutaML iteration through source model objects, by invoking the relevant content as part of the internal structure of the `lutaml_uml_datamodel_description` command through its `include_block` and `package_text` blocks.

So Figure 99 is excerpted from the LutaML command generating the CityGML conceptual model rendering as a LutaML data model description.

- Generating the conceptual model rendering involves iterating through the complete XMI file (`../../xmi-full/full-242.xml`), in the sequence given in `lutaml_packages.yml`, from “Core” to “Tunnel”.
- Before the packages are rendered, the Metanorma AsciiDoc ModSpec requirements with filenames starting with `requirements/requirements_class_` are iterated through, and included as initial content.
- Before each package is rendered, Metanorma AsciiDoc content is interpolated (`[.package_text, position="before", package="..."]`). That content is in fact the clause under which the UML package diagram and table appears: it includes a subclause heading, and prefatory text.
- After “Tunnel”, the Metanorma AsciiDoc thematic surface boundary files are iterated through, and included as embedded content,
- After that, another tranche of requirements is rendered (identified as filenames starting with `requirements/REQ_Classes_`, under the third level heading `==== Requirements`).

```
[lutaml_uml_datamodel_description,../../xmi-full/full-242.xmi,lutaml_packages.
yml]
---
[.include_block, index="1", position="before", base_path="requirements/
requirements_class_"]
....
....

[.package_text, index="2", position="before", package="Core"]
....
Unresolved directive in sources/sections/10-03-prototype-a1.adoc - include::
clause_8_2_core.adoc[]
....

(... other packages ...)

[.package_text, index="2", position="before", package="Tunnel"]
....
Unresolved directive in sources/sections/10-03-prototype-a1.adoc - include::
clause_8_18_tunnel.adoc[]
....

[.include_block, index="3", position="before", base_path="boundaries/
boundaries_"]
....
....

[.include_block, index="4", position="before", base_path="requirements/REQ_
Classes_"]
....
==== Requirements
....

(... other content ...)
```

```
---
```

Figure 99 — Instance of `lutaml_uml_datamodel_description` command in CityGML standard

Because the rendering styles were well-defined for what CityGML required, there was no need to customize the text within the command: CityGML does not use any Liquid templates to override the rendering of data parsed in the LutaML commands. All interpolated supplementary truth was dealt with straightforwardly as Metanorma AsciiDoc inclusions, typically at the file level.

It may prove necessary to do Liquid templating in future documents, to deal with parsed values explicitly; but consistent rendering of UML under OGC should be encouraged as more efficient.

LutaML artifacts are embedded in Metanorma as blocks, which can be cross-referenced with anchors.

As the `lutaml_uml_datamodel_description` command generates a sequence of artifacts, it is also possible to cross-reference specific generated artifacts:

- `lutaml_figure` provides a reference anchor to a figure defined in the XMI file, using its XMI ID or package for reference.

  Example 1: cross-references the UML diagram generated for UML package “Wrapper root package”, and adds the caption “Fig B1 Full model”.

- `lutaml_table` provides a reference anchor to the definition tables of a particular package, class, enumeration or data type object in the XMI.

  Example 2: cross-references the attributes table for UML class “my name” within UML package “Wrapper root package”.

  Example 3: The `lutaml_table` command is used several times in the CityGML standard to cross-reference other package definitions; for instance in OGC 20-010 Clause 6:

```
Classes painted in yellow belong to the Requirements Class
which is subject of discussion in that clause of the standard
in which the UML diagram is given. For example, in the
context of lutaml_table::[package="Core"], which introduces
the _CityGML Core_ module, the yellow color is used to denote
classes that are defined in the _CityGML Core_ Requirements
Class. Likewise, the yellow classes shown in the UML diagram
in lutaml_table::[package="Building"] are associated with the
_Building_ Requirements Class that is subject of discussion
in that chapter.
```

10.4.7. Rendering

Once the Metanorma AsciiDoc source is assembled out of its component truths, it can then be rendered using Metanorma into a number of output formats (https://www.metanorma.org/author/topics/building/output-formats/):

- Metanorma Semantic XML, capturing the structure and meaning of the standards document, and following the document model in ISO/AWI 36100.

- Metanorma Presentation XML, denormalizing the structure of the standards document in preparation for rendering, including resolving cross-references and generating auto-numbering.
- HTML (Table 8)
- PDF (Table 8)
- Microsoft Word

The rendering of documents into a Metanorma flavor specific to an SDO, such as OGC, conforms to the styles, conventions and branding set by that organization.

Metanorma provides a command-line executable, `metanorma`, that supports multiple commands.

The `compile` command is used for processing a single source document, and is also the default command if no command is invoked by default.

Example — Example of running the `metanorma compile` command: This command compiles the Metanorma AsciiDoc file `my-ogc-standard.adoc` into an HTML document, using the OGC flavor of Metanorma AsciiDoc.

```
# Explicit invocation of the `compile` command.
$ metanorma compile --type ogc -x html my-ogc-standard.adoc
# Omission of the `compile` command leads to `compile`, too.
$ metanorma --type ogc -x html my-ogc-standard.adoc
```

In practice, repositories may contain multiple Metanorma documents. Metanorma provides the `site generate` command that allows one to invoke the `metanorma` executable once while processing all desired documents. To specifically control what documents will be processed, one would need to add a site manifest file `metanorma.yml` for the entire repository.

NOTE: Further details on the `site generate` command can be found here: (https://www.metanorma.org/blog/2020-12-19-generate-mini-site-with-metanorma-cli/).

In the case of CityGML, only one file is involved, so the site manifest file is short (Figure 100):

```
---
metanorma:
  source:
    files:
      - sources/standard/20-010.adoc

  collection:
    organization: "OGC"
    name: "OGC TB 17 D144 CityGML XMI model-driven standard"
```

Figure 100 — OGC 20-010 Metanorma site manifest

10.5. Prototype A2: DGGS PIM to Standard derivation (OGC, ISO)

10.5.1. General

The DGGS conceptual model was originally developed for the MDS approach — the primary contributor of the DGGS models, Robert Gibb, developed the conceptual model tailored to the intended structure of the resulting standard.

The selection of the DGGS conceptual model in the prototype therefore provides good contrast against the CityGML 3.0 conceptual model as the latter was not intended for the MDS approach.

The DGGS conceptual model published in the form of a single EAP file by the ISO/TC 211 HMMG, under the package “ISO 19170-1 Conceptual Model”.

The published EAP file was evaluated for its readiness as a platform-independent Conceptual Model for use in the derivation of platform-specific Implementation Schema standards.

The original EAP file was used to generate the MDS and is used for all PIM-to-PSM work. While issues were identified, the original model was unchanged.

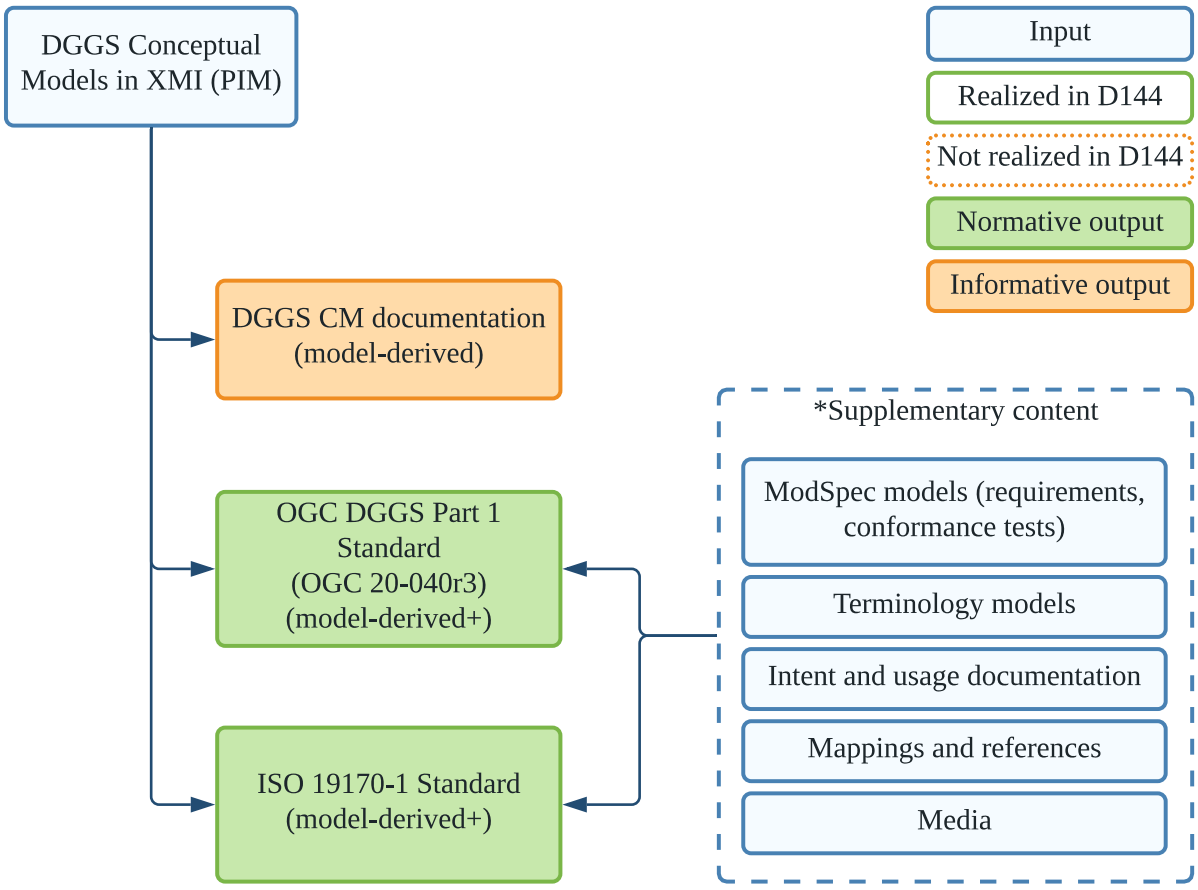


Figure 101 — Standards and deliverables derived from the DGGS conceptual models

The prototype is implemented and open-sourced at: https://github.com/metanorma/ogc-dggs-xmi (location change pending).

10.5.2. Previous workflow

The DGGS standard was previously dual-published as OGC 20-040 and ISO 19170, both generated via Metanorma.

The original standard was already authored as a MDS, where information was extracted from the conceptual model as managed on Sparx Systems Enterprise Architect.

However, the workflow that published the initial MDS documents was intensive and manual, as shown in Figure 102. It involved several steps and tools that required heavy manual processing, including:

- Using Microsoft Word to export Sparx Systems Enterprise Architect generated EMF images into the proper EMF format
- Exporting HTML from Sparx Systems Enterprise Architect by using a custom RTF template

- Writing custom Python scripts to convert Sparx Systems Enterprise Architect generated RTF annotations into HTML
- Incorporating the extracted HTML text into Metanorma AsciiDoc

This prototype aimed to fully generalize and streamline the workflow so that it can also be used by other authors in OGC for generating PIM-to-MDS output.

10.5.3. Methodology

In this prototype, the same methodology was followed as for CityGML:

- Export: Making the PIM available for processing (Clause 10.5.4)
- Authoring: Making the supplementary truth available for processing (Clause 10.5.5)
- Data parsing: Parsing the truth of the model into derived truth in the document (Clause 10.5.6)
- Integrating: Merging derived and supplementary truth into the target document (Clause 10.5.7)
- Rendering: Generating human-consumable presentations of the target document (Clause 10.5.8)

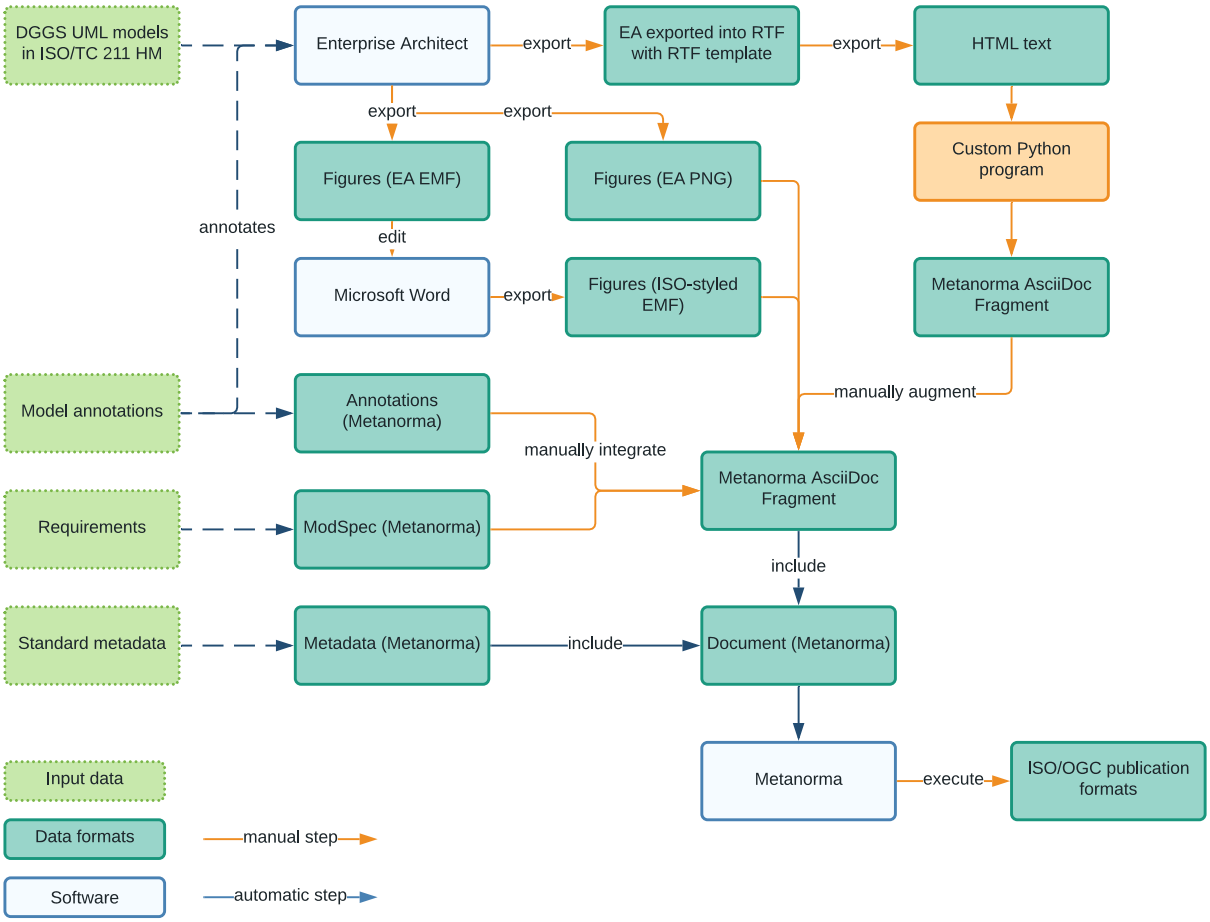


Figure 102 — Model-based standard generation flow with normal Metanorma

In the process of working on task D144, the workflow was substantially automated through the inclusion of the LutaML tool, to process the XMI contents automatically and render correct Metanorma AsciiDoc annotations.

Tooling was also added to Metanorma to process Sparx Systems Enterprise Architect generated EMF correctly, instead of relying on manual postprocessing.

The improved workflow is shown in Figure 103, and runs end-to-end without human intervention, which can be done in cloud systems via continuous integration.

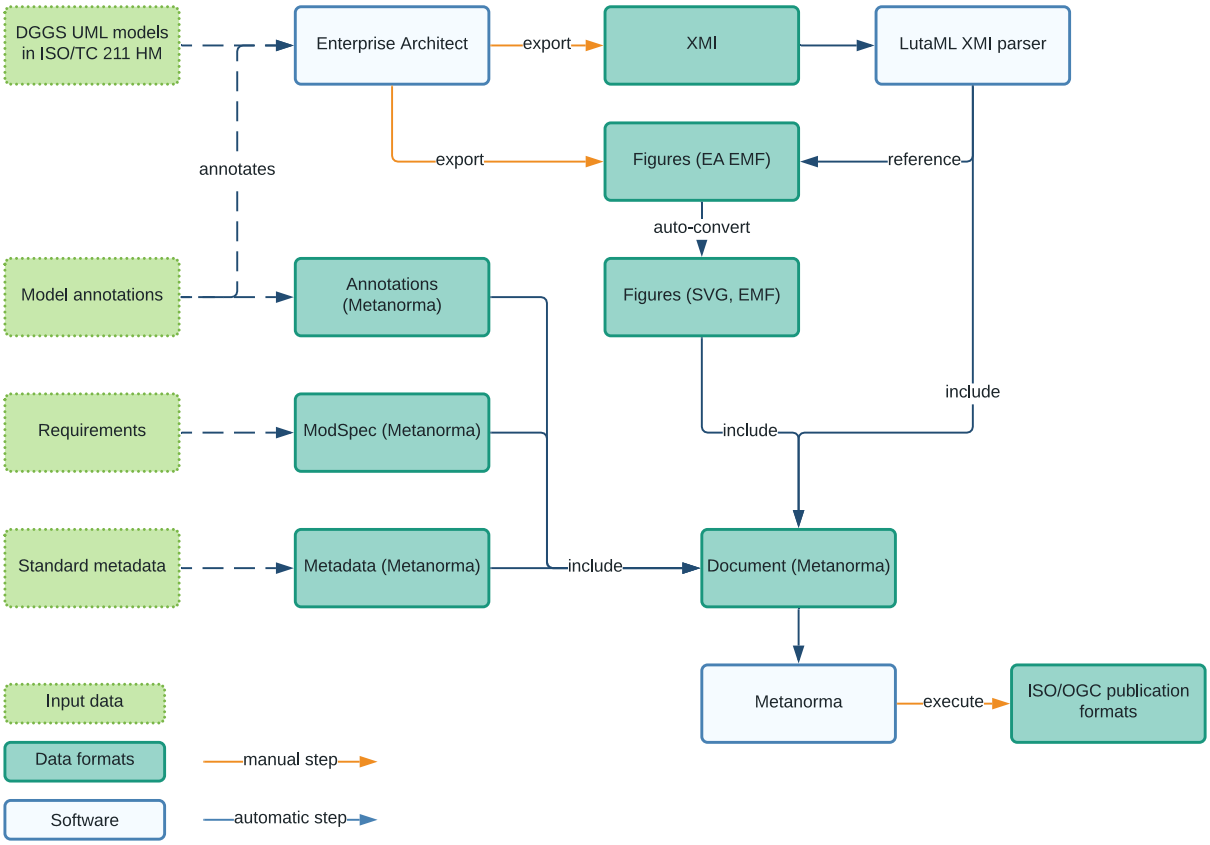
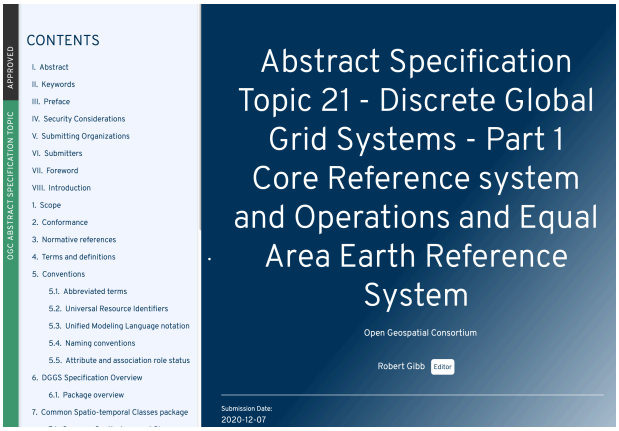


Figure 103 — Model-based standard generation flow managed by Metanorma MDS suite

As a result, the DGGS standard has now been published as OGC 20-040r3, in September 2021 Table 9.

Table 9 — OGC 20-040r3: DGGS standard in HTML and PDF formats



OGC 20-040r3: DGGS standard in HTML format

OGC 20-040r3: DGGS standard in PDF format

The prototype is implemented and open-sourced at: https://github.com/metanorma/ogc-dggs-xmi (location change pending).

10.5.4. Export

Making source data available for MDA involves exporting the information models in a standardized interoperable format from the model authoring tool.

The main truth for the DGGS conceptual model is the set of XMI files describing the DGGS UML models from the ISO/TC 211 Harmonized Model. The initial step in processing is to export those XMI files.

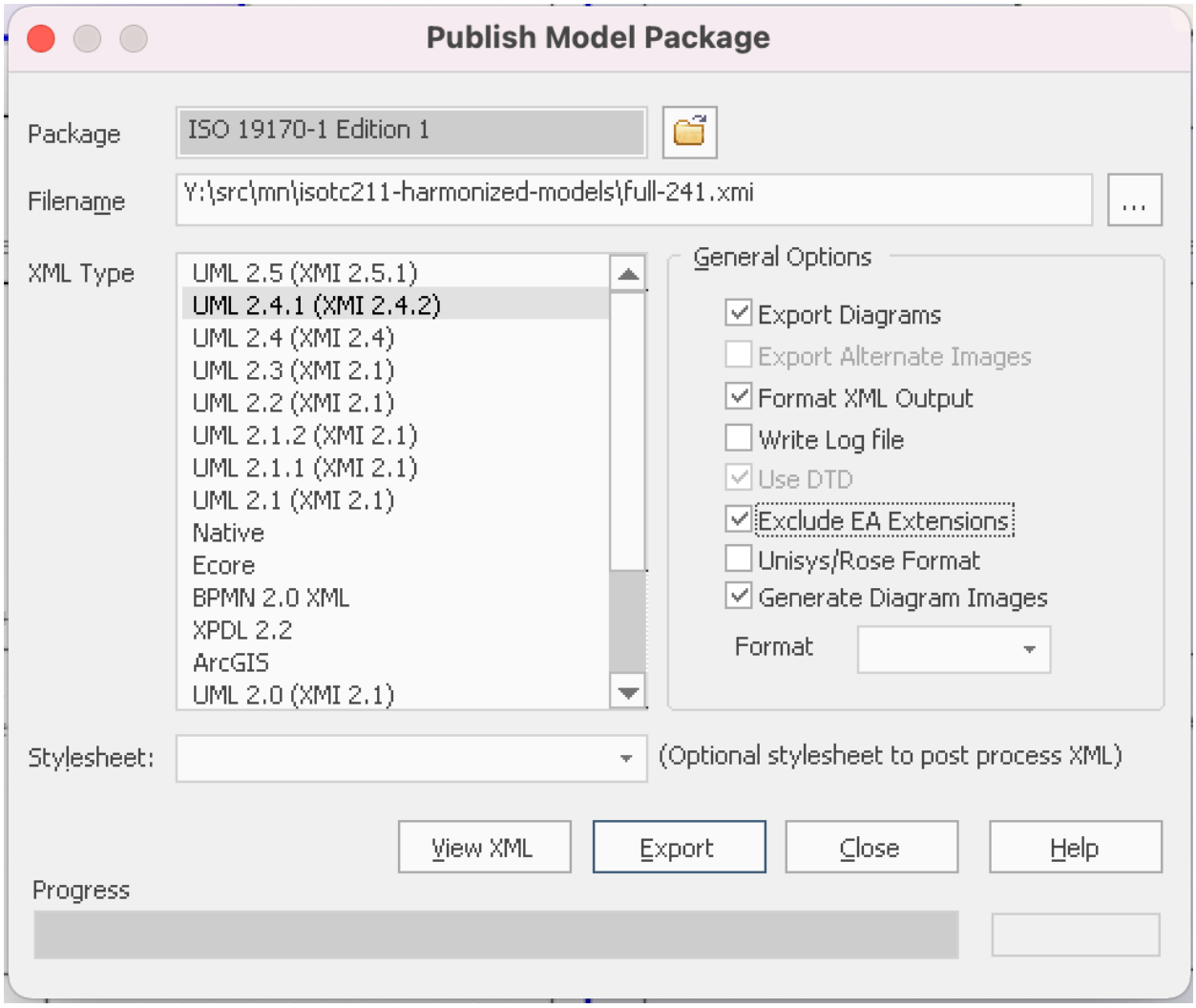


Figure 104 — Export options for DGGS EA model to XMI for Metanorma processing

Similar to that of Clause 10.4.3, the exported diagrams are used in the resulting MDS.

The other steps mirror those described in Clause 10.4.3.

10.5.5. Authoring

Similar to that of Clause 10.4.4, supplementary information was used as input to the DGGS MDS.

There are several differences in the representation of ModSpec instances:

1. The encoding of the ModSpec Conformance Class is different:

   - In CityGML 3.0 (Clause 10.4.4), a ModSpec “Conformance Class” is not encoded in the ModSpec instance format — they were implemented as individual sections.
   - In DGGS, the ModSpec “Conformance Class” objects are also not encoded in the proper ModSpec format, but as tables.

2. The contents of Clause 2 “Conformance” contain different types of information.

   - The CityGML 3.0 “Conformance” clause provides basic information about conformance targets, conceptual models, implementation specifications and conformance classes
   - The DGGS “Conformance” clause provides a listing of requirements belonging to each of the different Conformance Classes. However, this listing merely repeats what is already in its Annex A (“Abstract Test Suite”)

3. The title of the “Abstract Test Suite” differs:

   - The CityGML 3.0 document had an Annex A “Abstract Test Suite”
   - The DGGS document had an Annex A “Conformance Class Abstract Test Suite”

A.2. Conformance Class Core

ABSTRACT TEST A.1	
/ats/core/classes	
Requirement	/req/core/classes
Test purpose	To validate that the Implementation Specification correctly implements the UML Classes defined in the Conceptual Model.
Test method type	Manual Inspection
Test method	1. For each UML class defined or referenced in the Tunnel Package: a) Validate that the Implementation Specification contains a data element which represents the same concept as that defined for the UML class. b) Validate that the data element has the same relationships with other elements as those defined for the UML class. Validate that those relationships have the same

Figure 105 — CityGML 3.0 Conformance Class representation

A.1. Common Spatio-temporal Classes package conformance categories

The requirement tests that apply to each conformance category in the Common Spatio-temporal Classes package are listed in Table A.1

Table A.1 — Conformance classes for Common Spatio-temporal Classes package

Conformance class	Requirement tests
Temporal geometry and topology	Annex A.1, Requirement test A.1
Temporal reference systems using period identifiers	Annex A.1, Requirement test A.4
Spatial zone geometry and topology	Annex A.1, Requirement test A.2

Figure 106 — DGGS Conformance Class representation

2 CONFORMANCE

This Standard defines a *Conceptual Model* (Clause 4.1.6) which is independent of any encoding or formatting techniques. The *Standardization Targets* (Annex B.1.1.14) for this Standard are:

1. *Conceptual Models* (Clause 4.1.6) (extended versions of this conceptual model)
2. *Implementation Specifications* (Clause 4.1.8) (encodings of this conceptual model)

2.1. Conceptual Models

A Conceptual Model standardization target is a version of the CityGML 3.0 Conceptual Model (CM) tailored for a specific user community. This tailoring can include:

1. Omission of one or more of the optional UML packages;

Figure 107 — CityGML 3.0 Conformance clause (excerpt)

2 CONFORMANCE

This standard defines the conformance classes listed in tables Table 1–Table 3

Table 1 — Conformance classes for Common Spatio-temporal Classes package

Conformance class	Requirement tests
Temporal geometry and topology	Annex A.1, Requirement test A.1
Temporal reference systems using period identifiers	Annex A.1, Requirement test A.4
Spatial zone geometry and topology	Annex A.1, Requirement test A.2

Figure 108 — DGGS Conformance clause (excerpt)

This leads to two recommendations:

1. OGC should harmonize the encoding of ModSpec instances, including Conformance Class, such that they display identically across documents to minimize confusion.

2. OGC should standardize the type of content provided in Clause 2 “Conformance”, and standardize the titles for these mandatory sections, including “Conformance” and “Abstract Test Suite”.

10.5.6. Data parsing

The parsing step was much more straightforward for DGGS, as the iteration through the UML classes of the source data only needed to be done once for the standard, and there was no need to reorder or filter out packages from the DGGS conceptual model.

To transform the DGGS models into an MDS, the “automated inclusion” approach was used as the generated MDS document is intended to share the same order, hierarchy and arrangement of the UML models within UML packages specified in the model file.

This approach works for the DGGS MDS as the model itself reflects the intended MDS content order. The result is a very minimal but effective interface between the MDS document and the PIM model, with little to no need to customize model navigation behavior.

This approach places minimal burden on the MDS author when working with a PIM, since information available in the PIM are faithfully converted into elements in the MDS. An example is provided here with Figure 109 to demonstrate how data elements in the conceptual model are mapped to the MDS.

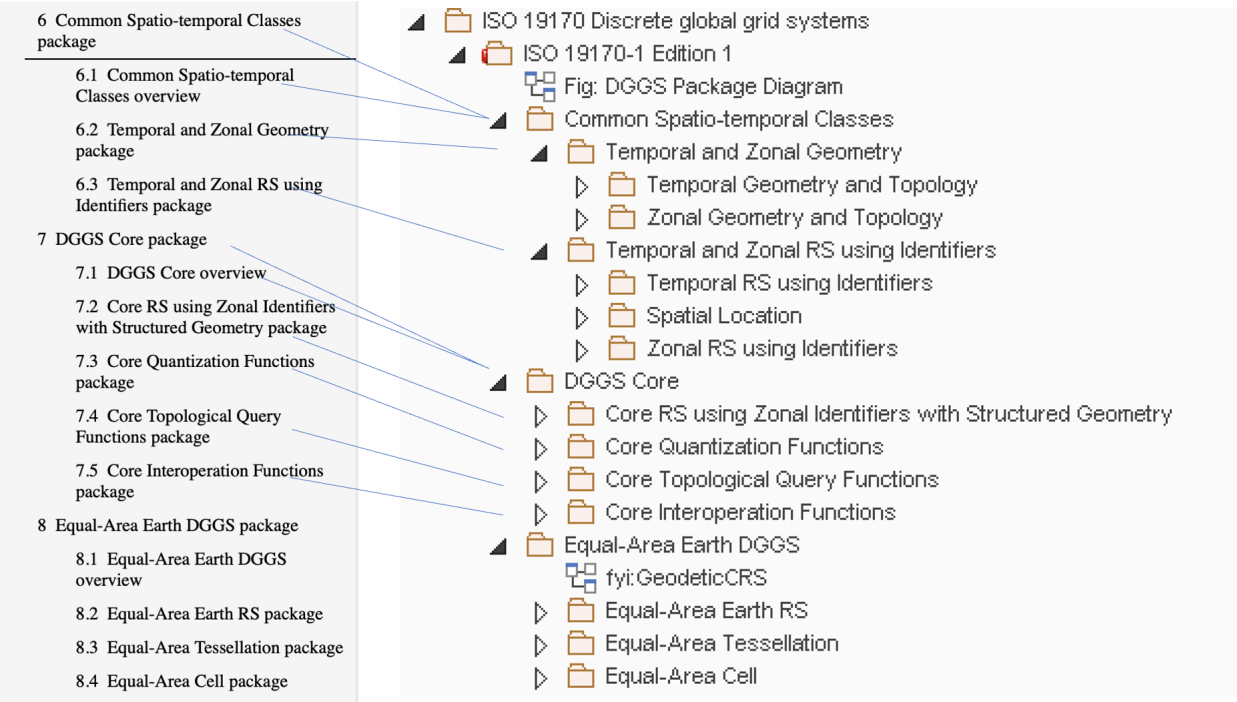


Figure 109 — Mapping between Table of Contents of ISO 19170/OGC 20-040r3 to DGGS conceptual model elements

The automatic inclusion behavior is implemented by the `lutaml_uml_datamodel_description` command (Figure 110) with a YAML configuration file (Figure 111) using the following steps:

1. In every UML package of the model file, investigate every UML class
2. If the UML package contains diagrams (in the `xmi-19170-only/Images` folder), include them all before content generation.
3. If there is a requirements Metanorma AsciiDoc file that uses the current UML package's name, include the file as content before content generation.
4. Render the UML class into the default UML representation template (embedded in LutaML).

```
[lutaml_uml_datamodel_description,../../xmi-19170-only/iso-19170-uml241-xmi242.
xmi]
---
[.diagram_include_block, base_path="../../xmi-19170-only/Images"]
...
...

[.include_block, base_path="requirements/"]
...
...
```

```
---
```

Figure 110 — Automated inclusion through the `lutaml_uml_datamodel_description` command in the DGGS standard

```
---
section_depth: 2
package_root_level: 2
```

Figure 111 — YAML configuration for the `lutaml_uml_datamodel_description` command in the DGGS standard

The `section_depth` value of 2 specifies that the location of the `lutaml_uml_datamodel_description` command is at the second level of depth, used to maintain the hierarchy of generated AsciiDoc sections.

The `package_root_level` value of 2 specifies that the automatic inclusion iterative process with UML packages at depth 2 of the XMI.

10.5.7. Integrating

The MDS document was integrated in a similar way as described in Clause 10.4.6, except that as seen in Figure 110, that the code used to insert ModSpec model instances is very short.

10.5.8. Rendering

Similar to that of Clause 10.4.7, the Metanorma AsciiDoc source is assembled out of its component truths and rendered using Metanorma into the following formats:

- Metanorma Semantic XML
- Metanorma Presentation XML
- HTML (Table 9)
- PDF (Table 9)
- Microsoft Word

The DGGS MDS site manifest is as follows (Figure 112), which generates both OGC and ISO outputs:

```
---
metanorma:
  source:
    files:
      - sources/as21-dggs/20-040r3.adoc
      - sources/as21-dggs/iso-19170-1-is-en-sections.adoc
```

```
collection:
  organization: "OGC"
  name: "OGC TB 17 D144 DGGS XMI model-driven standard"
```

Figure 112 — OGC 20-040r3 Metanorma site manifest

10.6. Prototype B1: UML to GML / XML Schema

10.6.1. General

The CityGML 3.0 Conceptual Model is published in the form of a set of XMI files that were generated from a single Sparx Systems Enterprise Architect Project (EAP) file. Models upon which there are UML package dependencies are included. The published EAP file documents the same model (the XMI files were directly generated from it); it was used directly as input to this task. As documented in an earlier section, a revised CityGML 3.0 Conceptual Model (in the form of an EAP file) was used as input to the UML-to-GML model transformation in this task.

The OGC CityGML Standards Working Group (SWG) is completing work to formalize a corresponding CityGML 3.0 Implementation Specification for GML; that work is documented at: https://github.com/opengeospatial/CityGML-3.0Encodings/tree/master/CityGML . It includes a preliminary ShapeChange project configuration file: https://github.com/opengeospatial/CityGML-3.0Encodings/blob/master/CityGML/ShapeChangeConfigurationFile_CityGML_3.0.xml . That preliminary ShapeChange project configuration file was used as an initial configuration in this work; it was significantly revised as the work proceeded. The draft CityGML 3.0 XML Schema files were used as a reference encoding.

As documented in an earlier section, in coordination with revision of CityGML 3.0 Conceptual Model the current ShapeChange software baseline (v2.10) was extended with one new XML Schema-related rule: `rule-xsd-cls-basictype-list`. This rule is independent of the CityGML 3.0 Conceptual Model, addressing the more general topic of the modeling of lists of basic types in Application Schemas for an XML Schema PSM. The revised ShapeChange project configuration file is documented in Annex B.2. Comments are included there explaining the purpose for all key configuration elements and parameter values.

10.6.2. ShapeChange Input Configuration

10.6.2.1. Constraints

OGC 20-010, documents fifteen (15) constraints. Of these, twelve are OCL invariant constraints that apply to classes within CityGML packages; the remaining three are Text invariant constraints that apply to class `GM_Object` (ISO 19107:2019).

In order to load, validate, and then successfully process these constraints, despite the fact that some are not specified using OCL, the input process must be configured to ignore constraints that are text- rather than OCL-based (else they will generate processing errors).

This is accomplished as follows:

```
<parameter name="constraintLoading" value="enabled"/>
<parameter name="oclConstraintTypeRegex" value="OCL"/>
```

The use of OCL constraints is discussed in greater detail in Clause 8.2.12 and Annex F.

During target processing valid OCL constraints will be used to derive Schematron assertions.

10.6.2.2. Stereotypes

The CityGML 3.0 Conceptual Model uses four non-standard stereotypes, two of which are aliased (https://shapechange.net/get-started/config/input/#Stereotype_aliases) for use with ShapeChange as follows:

```
<stereotypeAliases>
  <!-- Establish CityGML-specific stereotypes as aliases for well-known
stereotypes. -->
  <StereotypeAlias wellknown="property" alias="Property"/>
  <StereotypeAlias wellknown="version" alias="Version"/>
  <!-- Stereotypes <<ObjectType>> and <<TopLevelFeatureType>> will be ignored.
-->
</stereotypeAliases>
```

Figure 113 — CityGML 3.0 non-standard stereotypes

ShapeChange ignores stereotypes that are unrecognized (see: https://shapechange.net/app-schemas/uml-profile/#Stereotypes) and have not been aliased to a stereotype that it does recognize (see: https://shapechange.net/resources/config/StandardAliases.xml). Thus, the remaining two non-standard stereotypes are ignored.

RECOMMENDATION 18
/rec/mbs/uml/stereotypes
Identify and document a set of UML stereotypes for consistent adoption and employment when developing OGC conceptual models. For each stereotype identify the implications of its use for derived encodings in common implementation environments (e.g., XML Schema and JSON Schema). Encourage publication, registration, and reuse of new stereotypes as an OGC resource.

10.6.2.3. Tagged Values

In addition to tagged values defined in either the ISO 19109:2015 or ISO 19136 profiles of UML, the revised CityGML 3.0 Conceptual Model uses several well-known ShapeChange tagged

values (see: https://shapechange.net/app-schemas/uml-profile/#Tagged_Values) applied to «BasicType» classes, in particular:

- rangeMaximum: Defines the maximum range of a basic XML Schema type; e.g., 3.
- rangeMinimum: Defines the minimum range of a basic XML Schema type; e.g., 1.

The use of these tagged values is discussed in greater detail in Clause 8.2.10.

Other tagged values are present with default values, but in the revised conceptual model their values are no longer significant (e.g., xsdEncodingRule).

The preliminary ShapeChange project configuration file injected a heretofore unknown tagged value:

```
<parameter name="addTaggedValues" value="itemType"/>
```

This tagged value was used to annotate certain cases of «BasicType» classes in coordination with revisions to ShapeChange source code. That mechanism was replaced by one new XML Schema-related rule (rule-xsd-cls-basictype-list); in consequence this tagged value is no longer created or employed.

RECOMMENDATION 19
/rec/mbs/uml/tags
Identify and document a set of UML tag definitions (“tagged values”) for consistent adoption and employment when developing OGC conceptual models. For each, identify with which stereotype it may be used and identify the implications of its use for derived encodings in common implementation environments (e.g., XML Schema and JSON Schema). Encourage publication, registration, and reuse of new tag definitions as an OGC resource.

The use of UML profiles, stereotypes, and tagged values is discussed in greater detail in Clause 8.2.7 and Clause 8.2.13.

10.6.2.4. Package Selection

OGC 20-010 is unusual in that, rather than specifying a single Application Schema, it specifies a set of seventeen (17) related Application Schemas. As a conceptual model normally encompasses only a single Application Schema, it is necessary to configure ShapeChange to process all UML packages that match a pattern based on their assigned XML namespaces (rather than simply identifying and processing the first package found with applied stereotype «ApplicationSchema»):

```
<parameter name="appSchemaNamespaceRegex" value="^http://www.opengis.net/
citygml/.*"/>
```

Other than these specific cases the input configuration employs parameter default values as documented at https://shapechange.net/get-started/config/input/

10.6.3. ShapeChange Transformation Configuration(s)

No transformations are employed.

10.6.4. ShapeChange Target Configuration

10.6.4.1. General

Other than the following specific cases the target configuration employs parameter default values as documented at https://shapechange.net/targets/xsd/

10.6.4.2. Schematron

Conversion from OCL to Schematron-based assertions may be performed on the basis of a ShapeChange-internal syntax representation of OCL expressions. The specifics of ShapeChange support for employing OCL-based constraints are documented at: https://shapechange.net/targets/xsd/extensions/ocl/ and https://shapechange.net/targets/xsd/extensions/ocl-conversion-to-schematron-xslt2-query-binding/

A standard configuration for employing this capability is specified; a SCH file will be generated corresponding to each XSD file whose encoded classes have associated OCL constraints ("dynamizer", "pointCloud", and "relief"):

```
<targetParameter name="schematronFileNameTemplate" value="[[SCHEMA_XSD_
BASENAME]].sch" />
<targetParameter name="segmentSchematron" value="true" />
<targetParameter name="schematronQueryBinding" value="xslt2" />
```

While the proposed XML Schema-based encoding of the CityGML 3.0 Conceptual Model does not include the specification and use of Schematron assertions for data validation, this configuration demonstrates the feasibility of doing so.

10.6.4.3. Documentation

The proposed XML Schema-based encoding of the CityGML 3.0 Conceptual Model uses XML Schema annotation elements to incorporate inline information captured in the EAP as "notes" attached to modeling elements. By default, ShapeChange will generate and populate such annotation elements when "notes" have been populated.

In some cases, particularly for large schemas, it is desirable to be able to generate an XML Schema encoding that is annotation-free, thus compact, more quickly parsed, and more efficient when used for instance document validation where the human-targeted inline information is not useful.

In other situations, it is valuable to include inline documentation, and perhaps also indicate where documentation was missing in the conceptual model. This flexibility is achieved by employing `rule-xsd-all-no-documentation`.

A standard configuration for employing this capability is specified as follows where inline documentation is enabled, a template specifying the format of the documentation is given (see: https://shapechange.net/targets/uml-model/#documentationTemplate and https://shapechange.net/targets/xsd/#documentationTemplate; here the ShapeChange descriptor "documentation", previously loaded from "notes" in the EAP, is used), and a string-value is specified for use when the EAP "notes" are empty).

```
<targetParameter name="includeDocumentation" value="true" />
<targetParameter name="documentationTemplate" value="[[documentation]]" />
<targetParameter name="documentationNoValue" value="[no value specified]" />
```

Figure 114 — Configuration to include inline documentation

10.6.4.4. Encoding Rules

The proposed XML Schema-based encoding of the CityGML 3.0 Conceptual Model is intended to be mainly upwards compatible with that specified in CityGML 2.1. As such it is based on the same set of encoding rules — those specified by ISO 19136:2007. That set is collectively referenced within ShapeChange as `iso19136_2007`; it is discussed in greater detail in Clause 7.1.5 and is documented online at: https://shapechange.net/targets/xsd/gml/

For the proposed CityGML 3.0 Conceptual Model XML Schema-based encoding eight additional rules are employed, as follows:

```
<EncodingRule name="citygml" extends="iso19136_2007">
  <rule name="rule-xsd-rel-association-classes"/>
  <rule name="rule-xsd-cls-basictype"/>
  <rule name="rule-xsd-cls-basictype-list"/>
  <rule name="rule-xsd-cls-union-asGroup"/>
  <rule name="rule-xsd-prop-initialValue"/>
  <rule name="rule-xsd-prop-constrainingFacets"/>
  <rule name="rule-xsd-all-no-documentation"/>
  <rule name="rule-xsd-pkg-schematron"/>
</EncodingRule>
```

Figure 115 — CityGML 3.0 additional encoding rules

The first rule is that discussed in the preceding section; it enables the GML 3.3-based transformation of UML association classes. The second to last rule suppresses the generation of XML Schema annotation elements (unless otherwise specified using target parameters). The last rule activates the Schematron-based encoding of OCL constraints. The other rules are discussed in Annex B.2, "ShapeChange configurations — UML to GML" and documented further at: https://shapechange.net/targets/xsd/extensions/

The CityGML 3.0 Conceptual Model generally, but inconsistently, populates the tagged value "xsdEncodingRule" for model elements with the value "citygml". In order to ensure a consistent

encoding it is desirable to supply a default value of “citygml” to all model elements for which a value is not otherwise specified.

```
<targetParameter name="defaultEncodingRule" value="citygml"/>
```

There are two classes that instead specify “notEncoded”: ID and XALAddress.

10.6.4.5. XML Namespaces

In order to integrate the XML Schema-based encodings of external schemas (e.g., through the use of `<xi:include>` elements) it is necessary to establish XML namespace identifiers. Those for commonly used XML schemas are available as reusable resources (see Clause 7.1.7), thus:

```
<xi:include href="http://shapechange.net/resources/config/StandardNamespaces.
xml"/>
```

In the present case XML namespace identifiers for external schemas not already established by `StandardNamespaces.xml` are also required, including the establishment of XML schema document locations for use in import statements. In the case of xAL we identify the document location as a public resource URL:

```
<XmlNamespace nsabr="xAL" ns="urn:oasis:names:tc:ciq:xal:
3"
location="http://docs.oasis-open.org/ciq/v3.0/cs02/xsd/default/xsd/xAL.xsd"/>
```

XML namespace identifiers for all CityGML schemas need to be established since these are not specified in the UML Model using package tagged values. It is also necessary to establish XML schema document locations for use in import statements. In the following configuration statements, relative locations dependent on the filesystem are established rather than, e.g., public resource URLs.

```
<XmlNamespace nsabr="app" ns="http://www.opengis.net/citygml/appearance/3.0"
  location="./appearance.xsd"/>
<XmlNamespace nsabr="brid" ns="http://www.opengis.net/citygml/bridge/3.0"
  location="./bridge.xsd"/>
<XmlNamespace nsabr="bldg" ns="http://www.opengis.net/citygml/building/3.0"
  location="./building.xsd"/>
<XmlNamespace nsabr="pcl" ns="http://www.opengis.net/citygml/pointcloud/3.0"
  location="./pointCloud.xsd"/>
<XmlNamespace nsabr="frn" ns="http://www.opengis.net/citygml/cityfurniture/3.0"
  location="./cityFurniture.xsd"/>
<XmlNamespace nsabr="grp" ns="http://www.opengis.net/citygml/cityobjectgroup/3.
0"
  location="./cityObjectGroup.xsd"/>
<XmlNamespace nsabr="con" ns="http://www.opengis.net/citygml/construction/3.0"
  location="./construction.xsd"/>
<XmlNamespace nsabr="core" ns="http://www.opengis.net/citygml/3.0"
  location="./cityGMLBase.xsd"/>
<XmlNamespace nsabr="dyn" ns="http://www.opengis.net/citygml/dynamizer/3.0"
  location="./dynamizer.xsd"/>
<XmlNamespace nsabr="gen" ns="http://www.opengis.net/citygml/generics/3.0"
  location="./generics.xsd"/>
<XmlNamespace nsabr="luse" ns="http://www.opengis.net/citygml/landuse/3.0"
  location="./landUse.xsd"/>
<XmlNamespace nsabr="dem" ns="http://www.opengis.net/citygml/relief/3.0"
  location="./relief.xsd"/>
```

```
<XmlNamespace nsabr="tran" ns="http://www.opengis.net/citygml/transportation/3.
0"
  location="./transportation.xsd"/>
<XmlNamespace nsabr="tun" ns="http://www.opengis.net/citygml/tunnel/3.0"
  location="./tunnel.xsd"/>
<XmlNamespace nsabr="veg" ns="http://www.opengis.net/citygml/vegetation/3.0"
  location="./vegetation.xsd"/>
<XmlNamespace nsabr="vers" ns="http://www.opengis.net/citygml/versioning/3.0"
  location="./versioning.xsd"/>
<XmlNamespace nsabr="wtr" ns="http://www.opengis.net/citygml/waterbody/3.0"
  location="./waterBody.xsd"/>
```

Figure 116 — CityGML 3.0 namespace configurations

10.6.4.6. Map Entries

When concept models specify the use of classes from external models (either as concept model superclasses or as the range of concept model properties) it becomes necessary to establish "mappings" for elements of the concept model whose target encoding will reference the encoding of the external-model element. Each mapping expresses how to encode a key (usually primitive) UML model element using a "native" encoding type.

Those for GML 3.2 (and above) schema encodings are available as reusable resources (see Clause 7.1.7), thus:

```
<xi:include href="http://shapechange.net/resources/config/StandardMapEntries.
xml"/>
```

The proposed CityGML 3.0 XML Schema-based encoding of classes requires ten additional map entries (see: https://shapechange.net/targets/xsd/#XsdMapEntry):

```
<!-- ISO 19103 -->
<XsdMapEntry type="URI" xmlType="anyURI" .../>
<!-- ISO 19109 -->
<XsdMapEntry type="AnyFeature" xmlElement="gml:AbstractFeature" .../>
<!-- ISO 19111 -->
<XsdMapEntry type="SC_EngineeringCRS" xmlElement="gml:EngineeringCRS" .../>
<XsdMapEntry type="SC_CRS" xmlAttributeGroup="gml:SRSReferenceGroup" .../>
<!-- ISO 19123 -->
<XsdMapEntry type="CV_DiscreteGridPointCoverage" xmlElement="gml:
RectifiedGridCoverage" .../>
<!-- ISO 19136 -->
<XsdMapEntry type="Code" xmlType="gml:CodeType" .../>
<XsdMapEntry type="doubleList" xmlType="gml:doubleList" .../>
<XsdMapEntry type="DoubleOrNilReasonList" xmlType="gml:
DoubleOrNilReasonListType" .../>
<XsdMapEntry type="MeasureOrNilReasonList" xmlType="gml:
MeasureOrNilReasonListType" .../>
<!-- OASIS xAL -->
<XsdMapEntry type="XALAddress" xmlType=" xAL:Address" .../>
```

Figure 117 — CityGML 3.0 additional map entries

For conciseness each entry has been abbreviated here; Annex B.2, "ShapeChange configurations — UML to GML" documents their complete syntax.

The proposed CityGML 3.0 XML Schema-based encoding utilizes an infrequently used capability of the ShapeChange XML Schema target to suppress the encoding of UML properties (see: https://shapechange.net/targets/xsd/#XsdPropertyMapEntry). It is assumed that whatever functionality was intended by the inclusion of the UML property in the conceptual model is "somehow" accomplished by the `<xi:include>` of external-schema encodings. How that is accomplished would need to be documented by the accompanying Implementation Specification.

In all there are five suppressed UML properties (all attributes) assorted across two classes, as follows:

```
<XsdPropertyMapEntry property="AbstractFeature::featureID"/>
<XsdPropertyMapEntry property="AbstractFeature::identifier"/>
<XsdPropertyMapEntry property="AbstractFeature::name"/>
<XsdPropertyMapEntry property="AbstractFeature::description"/>
<XsdPropertyMapEntry property="ImplicitGeometry::objectID"/>
```

Figure 118 — CityGML 3.0 suppressed UML properties

Examination of the proposed CityGML 3.0 XML Schema-based encoding reveals that the encoding types for classes `AbstractFeature` and `ImplicitGeometry` are specified as extensions of `gml:AbstractGMLType`, which is specified in GML 3.2 as:

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

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

Figure 119 — CityGML 3.0 encoding types for classes `AbstractFeature` and `ImplicitGeometry`

The implicit encoding for these five suppressed UML properties in the CityGML 3.0 Conceptual Model is given in Table 10.

Table 10 — XML Schema for Suppressed UML Attributes

UML ATTRIBUTE	XML ENCODING
`AbstractFeature::featureID`	`<xs:attribute>` "gml:id"

UML ATTRIBUTE	XML ENCODING
AbstractFeature::identifier	<xs:element> "gml:identifier"
AbstractFeature::name	<xs:element> "gml:name"
AbstractFeature::description	<xs:element> "gml:description"
ImplicitGeometry::objectID	<xs:attribute> "gml:id"

The format and use of <sc:xsdMapEntry> and <sc:xsdPropertyMapEntry> are documented at: https://shapechange.net/targets/xsd/#XSD_Map_Entries

10.6.5. ShapeChange Log Analysis and Outcome

ShapeChange (current development baseline, including the new XML Schema-related rule: rule-xsd-cls-basictype-list) was used to process the revised CityGML 3.0 Conceptual Model, as specified in a Sparx Systems Enterprise Architect Project (EAP) file, using the configuration described in the preceding sections and fully documented in Annex B.2, "ShapeChange configuration — UML to GML".

Twenty (20) files are generated, three of which consist only of Schematron assertions. The 17 XML Schema files are identical to the proposed CityGML 3.0 Conceptual Model XML Schema-based encoding excepting the differences documented in Section 5.5.2.8 of CityGML 3.0 — Lists of "basic type".

There were two categories of Warnings logged while the <input> process validated the revised CityGML 3.0 Conceptual Model. There were no anomalous log entries stemming from execution of the <TargetXmlSchema> process.

The first category of Warnings consist of those involving no longer appropriate specifications of property multiplicity in external conceptual models.

The warnings are:

```
W The multiplicity value of 'dimension' is neither a number nor a known string.
 '*' is used instead.
W ... Context: Class 'Model::ISO TC211::ISO 19103 All::ISO 19103:2005
Conceptual schema language::Basic Types::Primitive::Numerics::Vector'

W The multiplicity value of 'size' is neither a number nor a known string. '*'
is used instead.
W ... Context: Class 'Model::ISO TC211::ISO 19103 All::ISO 19103:2005
Conceptual schema language::Basic Types::Primitive::Text::CharacterString'

W The multiplicity value of 'n' is neither a number nor a known string. '*' is
used instead.
W ... Context: Class 'Model::ISO TC211::ISO 19109 All::ISO 19109:2015 Rules
for Application Schema::Examples::Temporal::HistoryOfBuildings::GenericFeature'
```

The first two warnings are a consequence of the approach used in ISO/TS 19103:2005 to representing contingent multiplicities in ISO/TS 19103:2005 primitive types consisting of arrays of elements (Vector and CharacterString).

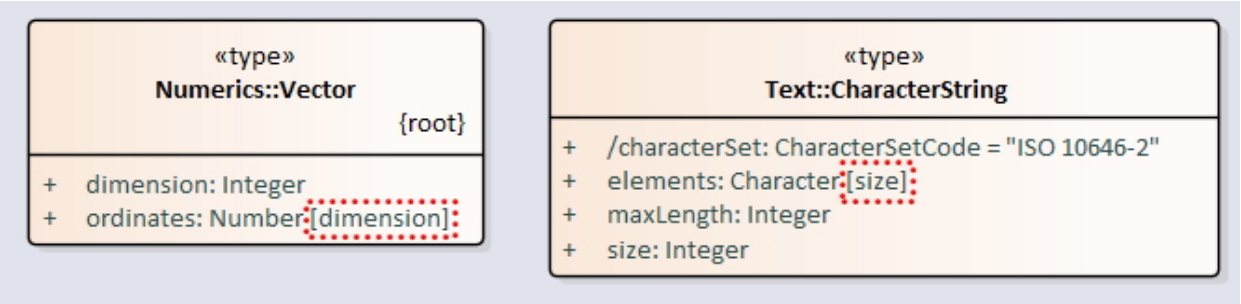


Figure 120 — ISO/TS 19103:2005 Arrays of Elements

These two classes were subsequently remodeled in ISO 19103:2015.

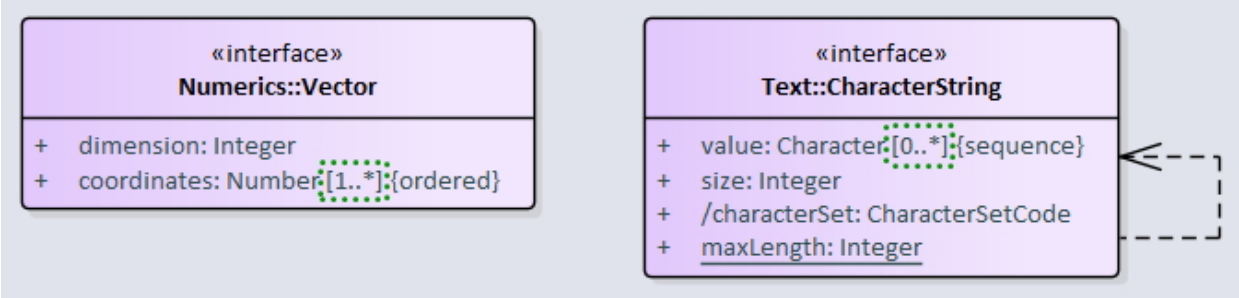


Figure 121 — ISO 19103:2015 Arrays of Elements

The third warning is a consequence of a model in the ISO 19109:2015 examples of feature temporality where an unlimited upper bound should have been indicated by * rather than n.

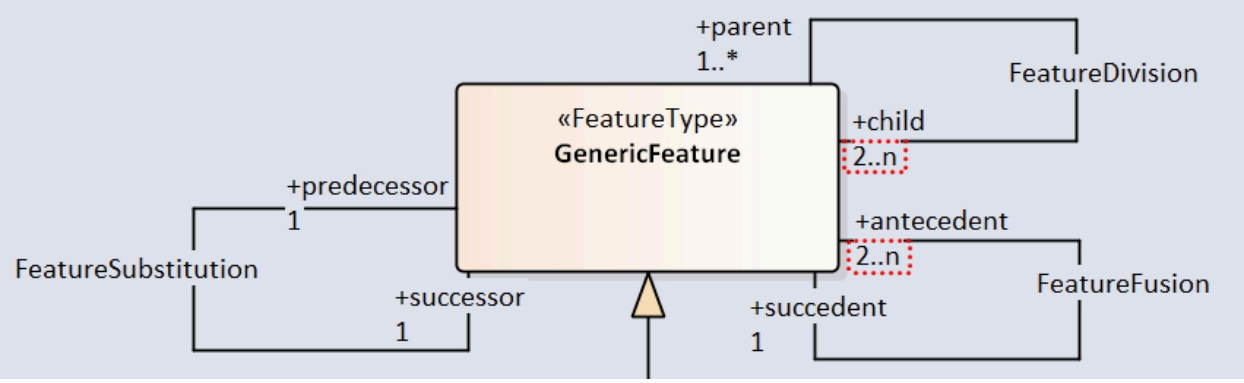


Figure 122 — ISO 19109:2015 Feature Temporality

These three warnings are not significant to the proposed CityGML 3.0 Conceptual Model XML Schema-based encoding.

The second category of Warnings are consistent with the definition that "AbstractSpaceBoundary is the abstract superclass for all types of space boundaries."

The warnings are:

```
W Restriction of property 'boundary' in class 'HollowSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'WaterBody' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BridgeRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BuildingRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'Storey' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstruction' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Door' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstructiveElement'
from supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractInstallation' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Window' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'TrafficSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AuxiliaryTrafficSpace' from
supertype 'AbstractSpace'.
```

The first case corresponds to the following class diagram:

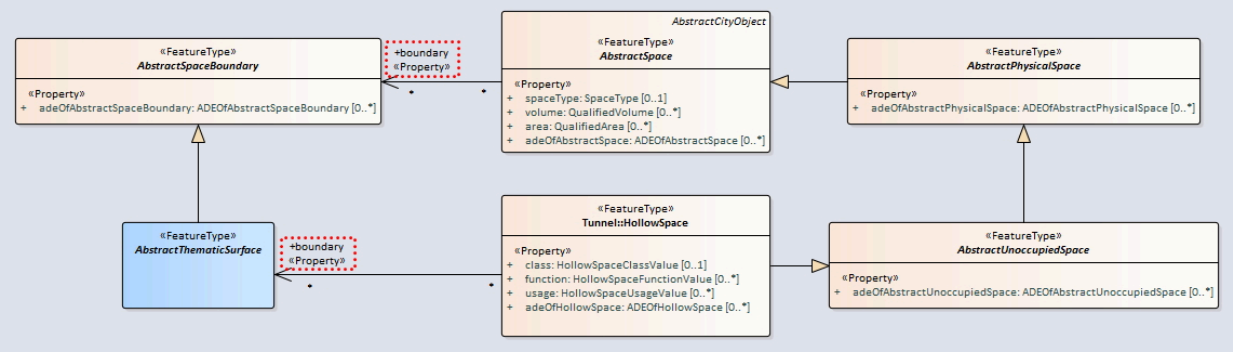


Figure 123 — Property `boundary` of `HollowSpace`

These warnings confirm the explicit modeling intent in the CityGML 3.0 Conceptual Model. In the resulting encoding the `boundary` element is assigned to the `AbstractSpace` rather than the `HollowSpace`, as follows:

```
<complexType name="AbstractSpaceType" abstract="true">
  <complexContent>
    <extension base="core:AbstractCityObjectType">
      <sequence>
        <element name="boundary" type="core:AbstractSpaceBoundaryPropertyType"
minOccurs="0" maxOccurs="unbounded"/>
        ...
      </sequence>
    </extension>
  </complexContent>
```

```
</complexType>
```

Figure 124 — CityGML 3.0 encoding of boundary

The choice of encoding the `boundary` property of `AbstractSpace` rather than that of HollowSpace may result in allowable range types for the `boundary` property with instances of HollowSpace that were not intended. Whether this allowance is significant to the proposed CityGML 3.0 Conceptual Model XML Schema-based encoding is uncertain.

10.7. Prototype B2: UML to JSON Schema

10.7.1. General

CityJSON version 1.1 is an implementation of the CityGML 3.0 conceptual model. The CityJSON JSON schema was meticulously hand-crafted in order to produce a minimal and lean application schema. This process of developing the CityJSON schema is also considered a PIM-to-PSM conversion process.

The OGC CityJSON version 1.0 Community Standard is a JSON-based encoding for storing 3D city models (see: OGC 20-072r2 and CityJSON) based on a subset of the CityGML 2.0 XSD-based data model (OGC 12-019). Its JSON Schemas were meticulously hand-crafted in order to produce a lean encoding; these schemas are published at: https://3d.bk.tudelft.nl/schemas/cityjson/

The OGC CityGML Standards Working Group (SWG) includes in its program of work the development of a complete JSON-based Implementation Standard derived from the CityGML 3.0 Conceptual Model — accompanied by potential harmonization with the OGC CityJSON Community Standard. GeoJSON and the work of the OGC Features and Geometries JSON SWG are expected to be taken into account in that activity.

The OGC UGAS-2020 Engineering Report (OGC 20-012) and the Application Schemas and JSON Technologies Engineering Report (OGC 18-091r2) describe mechanisms that have been implemented in the ShapeChange tool to generate JSON Schema specifications from UML-based conceptual models. Experience gained in those activities served as an important input to the work of the OGC Features and Geometries JSON SWG.

The CityGML 3.0 Conceptual Model is published in the form of a set of XMI files that were generated from a single Sparx Systems Enterprise Architect project (EAP) file. Models upon which there are UML package dependencies are included. The published EAP file documents the same model (the XMI files were directly generated from it); it was used directly as input to this task. As documented in an earlier section, a revised CityGML 3.0 Conceptual Model (in the form of an EAP file) was used as input to the UML-to-JSON Schema model transformation in this task.

As also documented in an earlier section, in coordination with revision of CityGML 3.0 Conceptual Model the current ShapeChange software baseline (v2.10) was extended with one new XML Schema-related rule: `rule-xsd-cls-basictype-list`.

This rule is independent of the CityGML 3.0 Conceptual Model, addressing the more general topic of the modeling of lists of basic types in Application Schemas for an XML Schema PSM. Although specific to the generation of XML Schemas, this rule indirectly affects model-preprocessing in ShapeChange and is thus significant to this task to generate JSON Schemas from the CityGML 3.0 Conceptual Model.

10.7.2. Model Preprocessing

ShapeChange includes the ability to use configuration parameters during model loading to overcome some categories of model shortfalls in important tagged values; this may be accomplished by either assigning default values uniformly or by supporting the configuration-determined addition of values to specific UML elements. At the present time these capabilities are principally limited to tagged values critical to the generation of XML Schemas.

Three pieces of information are required in order to control the specification of JSON Schema types (objects) and to organize the specifications of those objects into files.

These are:

- `jsonBaseURI` — Defines the base of the URI that is used as the value of the `id` field of a JSON Schema file generated by ShapeChange.
- `jsonDirectory` — Defines a subdirectory within the URI that is used as the value of the `id` field of a JSON Schema file generated by ShapeChange.
- `jsonDocument` — The name of the file (e.g., `MySchema.json`) in which the JSON Schema definitions for the classes contained the package will be encoded.

All types from a package in the model that is converted are stored in a single file. The URI designating that file is constructed as follows:

`<base-URI>/<subdirectory>/<document-name>`

Where:

- `<base-URI>` is the value of the `jsonBaseURI`; a tagged value `jsonBaseURI` defined on the package that contains the class takes precedence over the `jsonBaseURI` defined via the target configuration parameter.
- `<subdirectory>` is the value of the tagged value `jsonDirectory` defined on the package that contains the class; if that tagged value is undefined, the value of the `xmlns` tagged value is used; if that tagged value is also not defined, then the string `default` is used.
- `<document-name>` is the name of the file in which the JSON Schema definitions for the classes contained in the package will be encoded.

For the purposes of this task, it was determined that a single `<base-URI>` would be used for all packages within a standard (including together all of the Application Schemas that define the CityGML 3.0 Concept Model). This value was defined via a target configuration parameter.

The values for <subdirectory> and <document-name> were necessarily determined by setting tagged values on packages in the model.

The assignments made are listed in Table 11.

Table 11 — Assignments for tagged values in CityGML model

UML PACKAGE	JSONDIRECTORY	JSONDOCUMENT
CityGML::Appearance	ogc	Appearance_3.0.json
CityGML::Building	ogc	Building_3.0.json
CityGML::Bridge	ogc	Bridge_3.0.json
CityGML::Construction	ogc	Construction_3.0.json
CityGML::Core	ogc	Core_3.0.json
CityGML::Relief	ogc	Relief_3.0.json
CityGML::Dynamizer	ogc	Dynamizer_3.0.json
CityGML::CityFurniture	ogc	CityFurniture_3.0.json
CityGML::Generics	ogc	Generics_3.0.json
CityGML::CityObjectGroup	ogc	CityObjectGroup_3.0.json
CityGML::LandUse	ogc	LandUse_3.0.json
CityGML::PointCloud	ogc	PointCloud_3.0.json
CityGML::Transportation	ogc	Transportation_3.0.json
CityGML::Tunnel	ogc	Tunnel_3.0.json
CityGML::Vegetation	ogc	Vegetation_3.0.json
CityGML::Versioning	ogc	Versioning_3.0.json
CityGML::WaterBody	ogc	WaterBody_3.0.json
ISO TC211::ISO 19103 All	iso	ISO_19103_All.json

UML PACKAGE	JSONDIRECTORY	JSONDOCUMENT
ISO TC211::ISO 19108 All	iso	ISO_19108_All.json
ISO TC211::ISO 19109 All	iso	ISO_19109_All.json
ISO TC211::ISO 19111 All	iso	ISO_19111_All.json
ISO TC211::ISO 19123 All	iso	ISO_19123_All.json
ISO TC211::ISO 19136 All	iso	ISO_19136_All.json

10.7.3. Processing overview

Since the CityGML 3.0 Conceptual Model depends on conceptual schemas from other standards and those standards do not specify corresponding JSON Schemas, it is necessary to generate prototype JSON Schemas for those external conceptual schemas.

Therefore, there are two uses of ShapeChange required, as follows:

- ShapeChange is first used to generate JSON Schemas for external standards. One encoding (`.json`) file is generated for each standard. For each encoding file a ShapeChange map-entry file is generated that documents the JSON Schema type identifier corresponding to each UML class in the conceptual model.
- ShapeChange is then used to generate JSON Schemas for the seventeen application schemas composing the CityGML 3.0 Conceptual Model. The map-entry files generated in the first step are used to resolve references from CityGML properties to externally-defined classes.

The processing sequence is initiated by first defining encodings for key ISO 19103:2015 classes in terms of JSON primitives. ShapeChange will then be used to construct JSON Schema types based on those encodings.

Those map-entries (see: https://shapechange.net/targets/json-schema/#Map_Entries) are as follows:

```
<mapEntries>
  <MapEntry type="Boolean" rule="*" targetType="boolean" param=""/>
  <MapEntry type="Character" rule="*" targetType=
"string"
param="keywords{minLength=1;maxLength=1}"/>
  <MapEntry type="CharacterString" rule="*" targetType="string" param=""/>
  <MapEntry type="Date" rule="*" targetType=
"string"
param="keywords{format=date}"/>
```

```
  <MapEntry type="DateTime" rule="*" targetType=
"string"
param="keywords{format=date-time}"/>
  <MapEntry type="Duration" rule="*" targetType=
"string"
param="keywords{format=duration}"/>
  <MapEntry type="Time" rule="*" targetType=
"string"
param="keywords{format=time}"/>
  <MapEntry type="Decimal" rule="*" targetType="number" param=""/>
  <MapEntry type="Integer" rule="*" targetType="integer" param=""/>
  <MapEntry type="Number" rule="*" targetType="number" param=""/>
  <MapEntry type="Real" rule="*" targetType="number" param=""/>
  <MapEntry type="URI" rule="*" targetType=
"string"
param="keywords{format=uri}"/>
  <MapEntry type="URL" rule="*" targetType=
"string"
param="keywords{format=uri}"/>
  <MapEntry type="URN" rule="*" targetType=
"string"
param="keywords{format=uri}"/>
  <MapEntry type="GenericName" rule="*" targetType="string" param=""/>
  <MapEntry type="LocalName" rule="*" targetType="string" param=""/>
  <MapEntry type="MemberName" rule="*" targetType="string" param=""/>
  <MapEntry type="ScopedName" rule="*" targetType=
"string"
param="keywords{format=uri}"/>
</mapEntries>
```

Figure 125 — ShapeChange map-entries for JSON Schemas

Map-entries subsequently generated by ShapeChange for external conceptual models include additional information, for example:

```
<MapEntry type="Angle" rule="*"
  targetType="http://test.org/schema/iso/ISO_19103_All.json#/$defs/Angle"
  param="encodingInfos{entityTypeMemberPath=type}"/>
```

10.7.4. ShapeChange Input Configuration

10.7.4.1. Tagged Values

As noted in Clause 8.2.6, “External Conceptual Model Readiness”, the significant tagged value `xsdEncodingRule` was found to be irregularly populated with the value `iso19136_2007_INSPIRE_Extensions`.

No attempt was made to remove these values or to remove or modify any other existing tagged values for encoding rules in the EAP. Instead their use during the processing of external conceptual models was blocked, as follows:

```
<parameter name="ignoreEncodingRuleTaggedValues" value="true"/>
```

10.7.4.2. Constraints

OGC 20-010 documents fifteen (15) constraints. Of these, twelve are OCL invariant constraints that apply to classes within CityGML packages; the remaining three are Text invariant constraints that apply to class `GM_Object` (ISO 19107:2019). These constraints cannot be represented in JSON Schema and are not loaded.

10.7.4.3. Stereotypes

The CityGML 3.0 Conceptual Model uses five non-standard stereotypes, three of which are aliased (https://shapechange.net/get-started/config/input/#Stereotype_aliases) for use with ShapeChange as follows:

```
<stereotypeAliases>
  <!-- Establish CityGML-specific stereotypes as aliases for well-known
stereotypes. -->
  <StereotypeAlias wellknown="property" alias="Property"/>
  <StereotypeAlias wellknown="version" alias="Version"/>
  <StereotypeAlias wellknown="DataType" alias="BasicType"/>
  <!-- Stereotypes <<ObjectType>> and <<TopLevelFeatureType>> will be ignored.
-->
</stereotypeAliases>
```

Figure 126 — CityGML 3.0 non-standard stereotypes

ShapeChange ignores stereotypes that are unrecognized (see: https://shapechange.net/app-schemas/uml-profile/#Stereotypes) and have not been aliased to a stereotype that it does recognize (see: https://shapechange.net/resources/config/StandardAliases.xml). Thus, the remaining two non-standard stereotypes are ignored.

When processing external conceptual schemas no stereotype aliases are specified.

RECOMMENDATION 20
/rec/mbs/uml/label-stereotypes
Identify and document a set of UML stereotypes for consistent adoption and employment when developing OGC conceptual models intended to support Implementation Specifications. For each stereotype identify the implications of its use for derived encodings in common implementation environments (e.g., XML Schema and JSON Schema). Encourage publication, registration, and reuse of new stereotypes as an OGC resource.

10.7.4.4. Package Selection

The approach to specifying which portions of the EAP to process differ between the two uses of ShapeChange. In the case of external conceptual models:

- A regular expression match based on the package names is used to first filter the complete set of packages to the set of interest (see: https://shapechange.net/get-started/config/input/#appSchemaNameRegex).
- Then an exclusion filter is used to ignore three of the subordinate packages that contain classes (directly or in subordinate packages) that are not of interest (see: https://shapechange.net/get-started/config/input/#excludedPackages).
- Finally for the packages of interest it is necessary to ensure that information required for any application schema is specified, regardless of whether the package has the applied stereotype «Application Schema» (see: https://shapechange.net/get-started/config/input/#PackageInfo_definitions). This allows us to avoid further editing in the external conceptual models of tagged values required by the encoding rules.

The required configuration is as follows:

```
<parameter name="appSchemaNameRegex" value="ISO 191.*"/>
<parameter name=
"excludedPackages"
value="Collections,Examples,Multiplicities"/>
<packages>
  <PackageInfo packageName="ISO 19103 All" ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19103.xsd"/>
  <PackageInfo packageName="ISO 19108 All" ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19108.xsd"/>
  <PackageInfo packageName="ISO 19107 All" ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19107.xsd"/>
  <PackageInfo packageName="ISO 19109 All" ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19109.xsd"/>
  <PackageInfo packageName="ISO 19111 Referencing by
coordinates"
ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19111.xsd"/>
  <PackageInfo packageName="ISO 19123 All" ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19123.xsd"/>
  <PackageInfo packageName="ISO 19136 All" ns="https://example.org/schema/
iso"
nsabr="iso" xsdDocument="ISO 19136.xsd"/>
</packages>
```

Figure 127 — CityGML 3.0 package configuration

When processing the CityGML 3.0 Conceptual Model a simple regular-expression match based on the XML namespace (see: https://shapechange.net/get-started/config/input/#appSchemaNamespaceRegex) suffices, as follows:

```
<parameter name="appSchemaNamespaceRegex"
  value="^http://www.opengis.net/citygml/.*"/>
```

10.7.5. ShapeChange Transformation Configurations

10.7.5.1. Association Classes

The CityGML 3.0 Conceptual Model employs three UML association classes which cannot be directly represented in JSON Schema.

- CityObjectRelation relates AbstractCityObject instances to other AbstractCityObject instances
- Role relates CityObjectGroup instances to AbstractCityObject instances
- TextureAssociation relates ParameterizedTexture instances to AbstractTextureParameterization instances

A commonly employed ShapeChange transformation is the Association Class Mapper (see: https://shapechange.net/transformations/association-class-mapper/) which maps a UML association class into a semantically equivalent UML class and pair of associations, as defined by ISO 19136-2:2015, “Geographic information — Geography Markup Language (GML) — Part 2: Extended schemas and encoding rules (GML 3.3)”.

Clause 6.2 includes class diagrams illustrating the transformation.

This transformation is employed to replace all UML association classes in the CityGML 3.0 Conceptual Model with “simple” classes having revised properties. It is also employed when processing external conceptual models.

Note that the ShapeChange XML Schema target configuration for processing the CityGML 3.0 Conceptual Schema (see Annex B) includes `rule-xsd-rel-association-classes` which accomplishes the same result.

10.7.6. ShapeChange JSON Target Configurations

10.7.6.1. XML Schema Target

Both the external-model and CityGML-model ShapeChange processing configurations include an XML Schema target. This target is included in the configuration as it indirectly affects model-preprocessing in ShapeChange in areas relevant to conceptual models being processed in this task (e.g., the handling of multiple inheritance); this is a legacy dependency within ShapeChange. The configuration produces no output aside from an empty folder hierarchy named “ignore”. The included “CityGmlXsdMapEntries.xml” file contains the same map-entries that are used when executing the UML-to-GML process.

```
<TargetXmlSchema inputs="INPUT" class="de.interactive_instruments.ShapeChange.
Target.XmlSchema.XmlSchema" mode="enabled">
  <targetParameter name="outputDirectory" value="./ignore" />
  <targetParameter name="skipXmlSchemaOutput" value="true" />
  <targetParameter name="defaultEncodingRule" value="citygml" />
  <rules>
    <EncodingRule name="citygml" extends="iso19136_2007">
      <rule name="rule-xsd-cls-basictype"/>
      <rule name="rule-xsd-cls-basictype-list"/>
      <rule name="rule-xsd-cls-union-asGroup"/>
    </EncodingRule>
  </rules>
  <xi:include href="http://shapechange.net/resources/config/StandardNamespaces.
xml" />
  <xi:include href="./CityGmlXsdMapEntries.xml" />
</TargetXmlSchema>
```

Figure 128 — CityGML 3.0 map-entry for XML Schema target

10.7.6.2. JSON Schema Target

Both the external-model and CityGML-model ShapeChange processing configurations include a JSON Schema target. These two ShapeChange processing configurations share the following characteristics: use of the “2019-09” draft JSON Schema specification, a common base URI for specifying JSON object identifiers, and the name of the JSON member to be added to a JSON object in order to encode the type represented by that object (used with `rule-json-cls-name-as-entityType`).

```
<targetParameter name="jsonSchemaVersion" value="2019-09"/>
<targetParameter name="jsonBaseUri" value="http://test.org/schema"/>
<targetParameter name="entityTypeName" value="type"/>
```

10.7.6.3. Object Referencing

The ShapeChange processing configuration for external models specifies a default value for tagged value inlineOrByReference, in case that tagged value is undefined or has an empty value for the property (see: https://shapechange.net/targets/json-schema/#inlineOrByReferenceDefault). This may not be an optimal choice under all circumstances but does ensure maximum flexibility in the resulting JSON Schemas. The default JSON Schema encoding for the “by reference” case is “type”: “string”, “format”: “uri” (see: https://shapechange.net/targets/json-schema/#byReferenceJsonSchemaDefinition).

```
<targetParameter name="inlineOrByReferenceDefault"
  value="inlineOrByReference"/>
```

10.7.6.4. Map Entries

Both ShapeChange processing configurations produce ancillary map-entry files (see: https://shapechange.net/targets/json-schema/#writeMapEntries). For each encoding file a ShapeChange map-entry file is generated that documents the JSON Schema type identifier corresponding to each UML class in the conceptual model.

```
<targetParameter name="writeMapEntries" value="true"/>
```

As documented in Clause 10.7.3, encodings are defined for key ISO 19103 classes in terms of JSON primitives; both ShapeChange processing configurations include the applicable map-entries.

```
<mapEntries>
  <MapEntry type="Boolean" rule="*" targetType="boolean" param=""/>
  ...
</mapEntries>
```

The ShapeChange processing configuration for the CityGML 3.0 Conceptual Model includes the map-entry files generated by the preceding use of ShapeChange to generate JSON Schemas for external conceptual models.

```
<xi:include href="./schema/ISO_19103_All_mapEntries.xml"/>
<xi:include href="./schema/ISO_19107_All_mapEntries.xml"/>
<xi:include href="./schema/ISO_19108_All_mapEntries.xml"/>
<xi:include href="./schema/ISO_19109_All_mapEntries.xml"/>
<xi:include href="./schema/ISO_19111_Referencing_by_coordinates_mapEntries.
xml"/>
<xi:include href="./schema/ISO_19123_All_mapEntries.xml"/>
<xi:include href="./schema/ISO_19136_All_mapEntries.xml"/>
```

Figure 129 — CityGML 3.0 map-entry configuration for external conceptual models

10.7.6.5. Encoding Rules

ShapeChange defines a built-in defaultPlainJson encoding rule (see: https://docs.ogc.org/per/20-012.html#jsonschema_encodingrules_plainjson), which consists of five conversion rules, as follows:

- `rule-json-cls-name-as-anchor` — The names of UML classes are encoded as "$anchor", which is added at the start of the schema definition of the class (within the definitions schema). JSON Schema definitions that have an "$anchor" can be referenced using the plain text value of the anchor as fragment identifier, instead of using a more complex JSON Pointer.
- `rule-json-prop-derivedAsReadOnly` — The JSON Schema definition of a UML property that is derived will include the "readOnly" annotation with JSON value true.
- `rule-json-prop-initialValueAsDefault` — The JSON Schema definition of a UML attribute that has an initial value, is not owned by an enumeration or code list, and whose value type is mapped to "string", "number", or "boolean", will include the "default" annotation with that value.
- `rule-json-prop-readOnly` — The JSON Schema definition of a UML property that is read only or fixed will include the "readOnly" annotation with JSON value true.
- `rule-json-prop-voidable` — The JSON Schema definition of a UML property with stereotype «`voidable`», or with tagged value `nillable = true`, is defined in a way that only allows either a null value or a(n array of) actual value(s).

Both ShapeChange processing configurations extend the `defaultPlainJson` encoding rule set with four additional conversion rules:

- `rule-json-cls-basictype` — If a direct or indirect supertype of an application schema class is mapped to one of the simple JSON Schema types string, number, integer, or boolean, then it is treated as a "basic type" and mapped to a simple JSON Schema type. A basic type can be restricted using a number of JSON Schema keywords.
- `rule-json-cls-union-propertyCount` — Unions are converted to the JSON Schema definition of a JSON object where each union option is represented as an optional member of the JSON object. The choice between the options defined by the union is encoded using `"maxProperties" = "minProperties" = 1`.
- `rule-json-cls-codelist-uri-format` — All code lists are represented by a JSON Schema that restricts the type to "string", and states that the "format" is "uri". See also: `rule-json-cls-codelist-link`
- `rule-json-cls-name-as-entityType` — A new JSON member is added to the JSON object which represents the class that is being converted. The JSON member is required and string-valued. It should be used to encode the name of the type that is represented by the JSON object.

In the context of CityGML, application of `rule-json-cls-name-as-entityType` could be dropped. The use and potential value of including this additional JSON member is discussed in https://docs.ogc.org/per/20-012.html#jsonschema_schemaconversionrules_types_classname_typeidentification.

The resulting ShapeChange processing configuration is:

```
<targetParameter name="defaultEncodingRule" value="cityJsonISO"/>
<rules>
  <EncodingRule name="cityJsonISO" extends="defaultPlainJson">
    <rule name="rule-json-cls-basictype"/>
    <rule name="rule-json-cls-union-propertyCount"/>
    <rule name="rule-json-cls-codelist-uri-format"/>
    <rule name="rule-json-cls-name-as-entityType"/>
  </EncodingRule>
</rules>
```

Figure 130 — CityGML 3.0 ShapeChange processing configuration of encoding rules for JSON

Note that this set of encoding rules is independent of potential integration with the GeoJSON Schema.

10.7.7. ShapeChange GeoJSON Target Configurations

10.7.7.1. General

The CityGML 3.0 Conceptual Model could be encoded using JSON Schema in a manner identical to the encoding for externally-defined conceptual schemas (the cityJsonISO encoding rule in the preceding section), however it is worth exploring the degree to which the abilities and limitations of the GeoJSON Schema might be integrated with the CityGML 3.0 Conceptual Model encoding in JSON Schema.

10.7.7.2. Default Encoding Rule

ShapeChange defines a built-in defaultGeoJson encoding rule (see: https://docs.ogc.org/per/20-012.html#jsonschema_encodingrules_geojson), which consists of the five conversion rules in defaultPlainJson plus four additional rules, as follows:

- `rule-json-cls-defaultGeometry-singleGeometryProperty` — The geometry property of a class represents the default geometry, and is encoded as the top-level "geometry" member. If a class has multiple — potentially inherited — geometry properties with different names, none of them is selected as default geometry (because no informed choice can be made) and ShapeChange will log an error. See related `rule-json-cls-defaultGeometry-multipleGeometryProperties`.

- `rule-json-cls-ignoreIdentifier` — The identifier of a type with identity (feature type or object type) will be encoded using an identifier member that is provided by a common base type (e.g. the "id" member of a GeoJSON Feature, to which a generalization relationship exists for a given feature type — see `rule-json-cls-virtualGeneralization`), therefore no additional identifier property is created.

- `rule-json-cls-nestedProperties` — By default, the properties of a type with identity (feature type or object type) are converted to first-level properties of the resulting JSON object. In GeoJSON, feature properties are encoded within the GeoJSON "properties" member. Notable exceptions from that rule are the GeoJSON members "id", "geometry", and "bbox". This rule results in a JSON Schema that encodes the properties of a type with identity within a nested "properties" member — minus any properties that are mapped to the aforementioned GeoJSON keys.

- `rule-json-cls-virtualGeneralization` — Adds generalization relationships to specific kinds of classes — if a) the applicable ShapeChange JSON Schema target parameters have been set, and b) the class does not already have that generalization relationship via one of its supertypes. This rule enables encoding all classes with a certain stereotype with a common base type, the generalization relationship to such a base type being implied by the stereotype. In GML, for example, the common base type for classes with stereotype «`featureType`» is gml:AbstractFeature. Rather than explicitly modeling such a base type (e.g. AnyFeature defined by ISO 19109:2015), as well as explicitly modeling generalization

relationships to the base type, the encoding rule automatically adds that relationship to relevant schema types.

When employing `rule-json-cls-virtualGeneralization` the parameters baseJsonSchemaDefinitionForFeatureTypes (and possibly baseJsonSchemaDefinitionForObjectTypes) would be added to the configuration of the ShapeChange JSON Schema target with the value `https://geojson.org/schema/Feature.json`.

In this task two approaches to integrating with GeoJSON were explored: extending the CityGML 3.0 Conceptual Model with additional GeoJSON geometry types, and modifying the CityGML 3.0 Conceptual Model to "promote" selected geometric properties to become GeoJSON "geometry".

10.7.7.3. Extension Encoding Rule

In this approach the abstract class «FeatureType» `AnyFeature` is directly mapped to the JSON Schema `https://geojson.org/schema/Feature.json`. In consequence the JSON encoding of all subclasses are extended by the functionality specified by the GeoJSON Schema.

The result is identical to employing `rule-json-cls-virtualGeneralization` if it were the case that the CityGML 3.0 Conceptual Model did not include abstract class «FeatureType» `AnyFeature`, as illustrated in the following diagram:

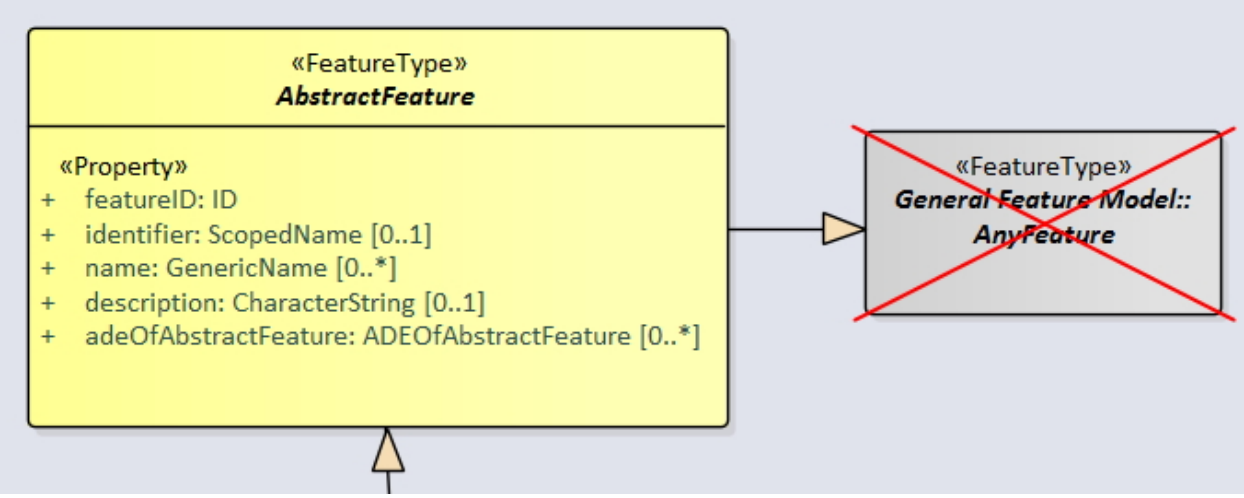


Figure 131 — Class AnyFeature use in CityGML 3.0 Conceptual Model

This enables the JSON encoding to employ the GeoJSON members `id`, `geometry`, and `bbox` in a conformant manner. `rule-json-cls-nestedProperties` ensures that the nested properties design of the JSON Schema is honored.

The resulting ShapeChange processing configuration is:

```
<rules>
  <EncodingRule name="cityJson">
```

```
    <rule name="rule-json-cls-nestedProperties"/>

    <!-- defaultPlainJson rules -->
    <rule name="rule-json-cls-name-as-anchor"/>
    <rule name="rule-json-prop-derivedAsReadOnly"/>
    <rule name="rule-json-prop-initialValueAsDefault"/>
    <rule name="rule-json-prop-readOnly"/>
    <rule name="rule-json-prop-voidable"/>

    <!-- extension rules -->
    <rule name="rule-json-cls-basictype"/>
    <rule name="rule-json-cls-union-propertyCount"/>
    <rule name="rule-json-cls-codelist-uri-format"/>
    <rule name="rule-json-cls-name-as-entityType"/>
  </EncodingRule>
</rules>

<mapEntries>
  ...
  <MapEntry type="AnyFeature" rule=
"*"
targetType="https://geojson.org/schema/Feature.json" param=""/>
</mapEntries>
```

Figure 132 — CityGML 3.0 encoding rule for `AnyFeature`

The generalization relationship from «FeatureType» AnyFeature is implemented by converting the class to a JSON Schema that consists of an "allOf" with two subschemas: the first being a "$ref" with the URI defined by the map-entry for AnyFeature, the second being the schema produced by applying the other conversion rules to the class (the only exception being rule-json-cls-name-as-anchor, because the $anchor created by that rule is not encoded in the second subschema, but in the schema that contains the allOf).

For example:

```
"AbstractFeature": {
  "$anchor": "AbstractFeature",
  "allOf": [
    {
      "$ref": "https://geojson.org/schema/Feature.json"
    },
    {
      "type": "object",
      "properties": {
        "properties": {
          "type": "object",
          "properties": {
            "type": {
              "type": "string"
            },
            "featureID": {
              "$ref": "http://test.org/schema/ogc/Core_3.0.json#ID"
            },
            "identifier": {
              "type": "string",
              "format": "uri"
            },
            "name": {
              "type": "array",
```

```
              "items": {
                "type": "string"
              },
              "uniqueItems": true
            },
            "description": {
              "type": "string"
            },
            "adeOfAbstractFeature": {
              "type": "array",
              "items": {
                "$ref": "http://test.org/schema/ogc/Core_3.0.
json#ADEOfAbstractFeature"
              },
              "uniqueItems": true
            }
          },
          "required": [
            "featureID",
            "type"
          ]
        }
      },
      "required": [
        "properties"
      ]
    }
  ]
}
```

Figure 133 — JSON Schema implementing «FeatureType» AnyFeature

Allowed values for the GeoJSON `geometry` member are currently limited to:

- https://geojson.org/schema/Point.json
- https://geojson.org/schema/LineString.json
- https://geojson.org/schema/Polygon.json
- https://geojson.org/schema/MultiPoint.json
- https://geojson.org/schema/MultiLineString.json
- https://geojson.org/schema/MultiPolygon.json
- https://geojson.org/schema/GeometryCollection.json

The added GeoJSON `geometry` member would then be populated with a simplified geometry value by a CityGML3.0 application — using rules to be defined by the CityGML Implementation Standard for the JSON encoding.

Note that per RFC 7946:

> The coordinate reference system for all GeoJSON coordinates is a geographic coordinate reference system, using the World Geodetic System 1984 (WGS 84) [WGS84] datum, with longitude and latitude units of decimal degrees. This is equivalent to the coordinate reference system identified by the Open Geospatial

Consortium (OGC) URN `urn:ogc:def:crs:OGC::CRS84`. An OPTIONAL third-position element SHALL be the height in meters above or below the WGS 84 reference ellipsoid. In the absence of elevation values, applications sensitive to height or depth SHOULD interpret positions as being at local ground or sea level.

Note that although the GeoJSON Schema requires the `geometry` member to be present, its value may be `null`.

From https://geojson.org/schema/Feature.json :

```
"required": [
  "type",
  "properties",
  "geometry"
],
...
  "geometry": {
    "oneOf": [
      {
        "type": "null"
      },
    ...
    ]
  }
```

The OGC Features and Geometries JSON SWG (see: https://github.com/opengeospatial/ogc-feat-geo-json/issues/23) considered adding a `where` member as an extended and extensible version of the value range of the `geometry` member that would include two more geometry types:

- http://docs.ogc.org/DRAFTS/21-045.html#_polyhedron
- http://docs.ogc.org/DRAFTS/21-045.html#_multipolyhedron

This possibility is discussed in greater detail in Section 6.9 of the OGC Testbed-17: OGC Features and Geometries JSON Engineering Report (OGC 21-017r1).

The new"`where` member would also support declaring the coordinate reference system of the coordinates of the geometry. This possibility is discussed in greater detail in the OGC Testbed-17: Features and Geometries JSON CRS Analysis of Alternatives Engineering Report (OGC 21-018).

Note that since 19 February 2022, the SWG has relabelled the 'where' element to 'place'.

10.7.7.4. Modification Encoding Rule

An alternative encoding strategy is to modify the CityGML 3.0 Conceptual Model to "promote" selected geometric properties to become GeoJSON `geometry` members. ShapeChange supports this strategy through the use of either `rule-json-cls-defaultGeometry-singleGeometryProperty` or `rule-json-cls-defaultGeometry-multipleGeometryProperties`.

In the singular case, a geometry property of a class represents the default geometry, and is then encoded as the top-level `geometry` member. If a class has multiple — potentially inherited — geometry properties with different names, none of them is selected as default geometry (because no informed choice can be made) and an error will be logged.

In the multiple case, a geometry property is identified as default geometry by setting tagged value defaultGeometry on the property to the value true. That property will then be encoded as the top-level “geometry” member. If multiple such properties exist (potentially inherited), none of them is selected as default geometry (because no informed choice can be made) and an error will be logged.

A sample of the resulting behavior can be explored by adding `rule-json-cls-defaultGeometry-singleGeometryProperty` to the existing configuration, and then revising file `ISO_19107_All_mapEntries.xml` as follows:

```
<!--
<MapEntry param="encodingInfos{entityTypeMemberPath=type}" rule="*"
targetType="http://test.org/schema/iso/ISO_19107_All.json#/$defs/GM_Point"
type="GM_Point"/>
-->
<MapEntry param="geometry" rule="*"
  targetType="https://geojson.org/schema/Point.json" type="GM_Point"/>
```

The encodings of three classes will be affected:

- `AbstractSpace` will gain a `geometry` member while losing the property `lod0Point`
- `ImplicitGeometry` will gain a `geometry` member while losing the property `referencePoint`
- `GeoreferencedTexture` will gain a `geometry` member while losing the property `referencePoint`

In all three cases the “geometry” member will be specified as:

```
"geometry": {
  "ref": "https://geojson.org/schema/Point.json"
},
```

In all cases the coordinate reference system will be limited to WGS 84 (OGC:CRS84 or OGC:CRS84h).

The same basic procedure could be followed for `GM_Surface` (used as the extent of `AbstractReliefComponent`) and `GM_MultiSurface` (used as the `lod0MultiSurface` of `AbstractThematicSurface`).

This modification-based approach to integrating GeoJSON was not pursued further as it seems likely that the direction being taken in draft JSON-FG will be more acceptable to the CityGML SWG and others.

10.7.8. ShapeChange Log Analysis and Outcomes

ShapeChange (current development baseline, including the new XML Schema-related rule: `rule-xsd-cls-basictype-list`) was used to process the revised and extended (see Clause 10.7.2) CityGML 3.0 Conceptual Model, as specified in a Sparx Systems Enterprise Architect Project (EAP) file, using the two configurations described in the preceding sections and fully documented in Annex B.3.

Three sets of files are generated and placed in the folder “schema”:

- Subfolder `iso` contains the seven (7) JSON Schemas for the external standards,
- Subfolder `ogc` contains the seventeen (17) JSON Schemas for CityGML 3.0, and
- Directly contained are the corresponding 24 JSON Schema map-entry files.

In the CityGML 3.0 Conceptual Model class `«DataType» XALAddress` is intended to operate as a surrogate for “address details” from the OASIS xAL standard (Figure 134).

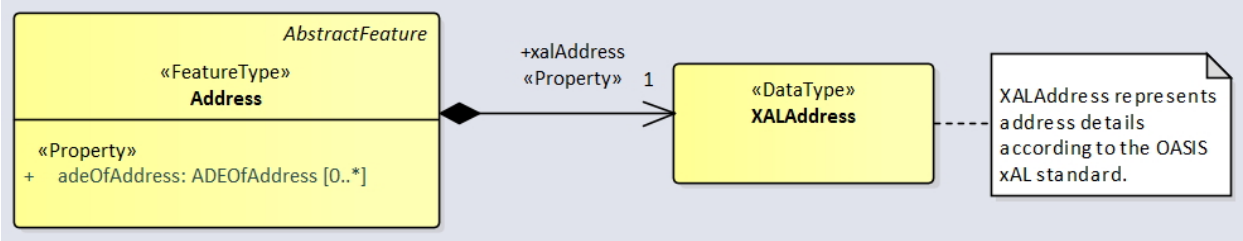


Figure 134 — CityGML 3.0 Part 1 — `«FeatureType» Address`

While in the case of XML Schema there exists an authoritative Implementation Specification (http://docs.oasis-open.org/ciq/v3.0/cs02/xsd/default/xsd/xAL.xsd), there is no corresponding authoritative Implementation Specification for JSON Schema. In consequence the current encoding treats the class as a new type, rather than a reference to an existing type from an external standard.

Thus:

```
"Address": {
  "$anchor": "Address",
  "allOf": [
    {
      "$ref": "http://test.org/schema/ogc/Core_3.0.json#AbstractFeature"
    },
    {
      "type": "object",
      "properties": {
        "properties": {
          "type": "object",
          "properties": {
            "xalAddress": {
              "$ref": "http://test.org/schema/ogc/Core_3.0.json#XALAddress"
```

```
            },
            ...
          },
          "required": [
            "xalAddress"
          ]
        }
      },
      "required": [
        "properties"
      ]
    }
  ]
},
"XALAddress": {
  "$anchor": "XALAddress",
  "type": "object",
  "properties": {
    "type": {
      "type": "string"
    }
  },
  "required": [
    "type"
  ]
}
```

Figure 135 — JSONSchema implementing «FeatureType» Address

When processing the external conceptual models there were both Errors and Warnings logged while the <input> process validated the models. There were no anomalous log entries stemming from execution of the <Target> process for JSON Schema.

The Errors consisted of twelve classes with more than one supertype (e.g., TM_TemporalCRS); multiple inheritance requires special consideration when developing technology-specific model implementations. At the present time ShapeChange arbitrarily ignored all but one supertype. The fidelity of the resulting JSON Schema with respect to the requirements of the CityGML 3.0 Conceptual Model will need to be evaluated.

The Warnings consisted of 70 cases in which a same-named property is specified on a superclass as on a class, for example:

```
W Restriction of property 'uom' in class 'Velocity' from supertype 'Measure'.
W Restriction of property 'surface' in class 'GM_Triangle' from supertype 'GM_
Polygon'.
W Restriction of property 'datum' in class 'SC_EngineeringCRS' from supertype
'SC_SingleCRS'.
```

This category of Warnings is discussed further alongside similar Warnings when processing the CityGML 3.0 Conceptual Model.

When processing the revised CityGML 3.0 Conceptual Model there was only a single category of Warnings logged while the <input> process validated the model. There were no anomalous log entries stemming from execution of the <Target> process for JSON Schema. These Warnings are consistent with the definition that “AbstractSpaceBoundary is the abstract superclass for all types of space boundaries.”

The warnings are:

```
W Restriction of property 'boundary' in class 'HollowSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'WaterBody' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BridgeRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BuildingRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'Storey' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstruction' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Door' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstructiveElement'
from supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractInstallation' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Window' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'TrafficSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AuxiliaryTrafficSpace' from
supertype 'AbstractSpace'.
```

The first case corresponds to the class diagram in Figure 136.

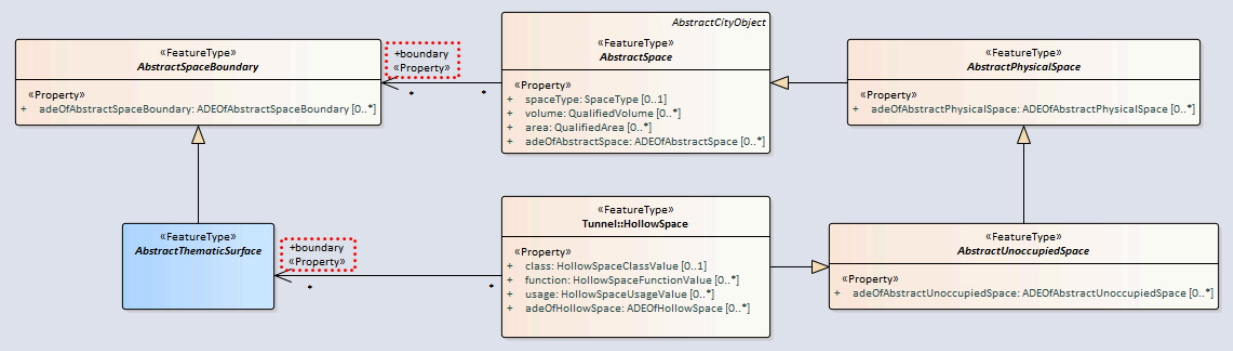


Figure 136 — Property "boundary" of HollowSpace

These warnings confirm the explicit modeling intent in the CityGML 3.0 Conceptual Model. In the resulting encoding the `boundary` element is assigned to the AbstractSpace rather than the HollowSpace, as follows:

```
"AbstractSpace": {
  "$anchor": "AbstractSpace",
  "allOf": [
    {
      "$ref": "http://test.org/schema/ogc/Core_3.0.json#AbstractCityObject"
    },
    {
      "type": "object",
      "properties": {
        "properties": {
          "type": "object",
          "properties": {
            "spaceType": {
              "$ref": "http://test.org/schema/ogc/Core_3.0.json#SpaceType"
```

```
            },
            ...
            "boundary": {
              "type": "array",
              "items": {
                "oneOf": [
                  {
                    "type": "string",
                    "format": "uri"
                  },
                  {
                    "$ref": "http://test.org/schema/ogc/Core_3.0.
json#AbstractSpaceBoundary"
                  }
                ]
              },
              "uniqueItems": true
            },
            ...
          }
        }
      }
    }
  ]
}
```

Figure 137 — JSON Schema implementing boundary

The choice of encoding the “boundary” property of AbstractSpace rather than that of HollowSpace may result in allowable range types for the “boundary” property with instances of HollowSpace that were not intended. Whether this allowance is significant to a CityGML 3.0 Conceptual Model JSON Schema-based encoding is uncertain.

10.8. Prototype B3: UML to RDF

10.8.1. General

OGC 18-032r2 describes a process for generating an application schema compliant with ISO 19150-2.

There is a known implementation that uses ShapeChange to generate RDF/XML and N-triple encodings. This was demonstrated in Testbed-14 and is documented in OGC 18-091r2. This process represents a PIM-to-PSM transformation.

https://github.com/opengeospatial/CityGML-3.0Encodings

NOTE: Work for this prototype is still ongoing.

A

ANNEX A (INFORMATIVE) UML-TO-GML APPLICATION SCHEMA ENCODING RULES

ANNEX A (INFORMATIVE) UML-TO-GML APPLICATION SCHEMA ENCODING RULES

A valid UML Application Schema conforms to the conceptual schema rules specified in ISO 19103:2015, the application schema rules specified in ISO 19109:2015, and the general encoding requirements of ISO 19136-1:2020. Those encoding requirements are summarized in the following list.

A.1. General

- The visibility of all UML elements shall be set to “public”. Only publicly visible elements shall be part of Application Schemas used for data interchange between applications.
- Documentation of the elements in the UML model shall be stored in tagged value “documentation”. Application schema:
- The UML Application Schema shall be represented by a package with the stereotype `«Application Schema»`.
    - This package shall contain (i.e., own directly or indirectly) all UML model elements to be mapped to object types in the GML application schema.
    - The package may include other packages without the stereotype `«Application Schema»` to group the different UML model elements within the application schema.
    - A unique XML namespace shall be associated with the UML Application Schema.
    - Tagged values “targetNamespace” for the target namespace URI and “xmlns” for the abbreviation shall be set if and only if the package represents a UML application schema.
    - The version number of a package representing a UML Application Schema shall be specified in a tagged value “version”, if applicable.

- The UML model shall be complete and not contain external references unless:
  - Predefined classes are imported from the standardized schemas of the ISO 19100 series of International Standards.
  - The classes from the ISO 19100 series of International Standards that are implemented by the GML schema and used by the UML application schema are specified in a package with the name “ISO19100” or any sub-package of a package with that name.

A.2. Packages

- If a package shall be mapped to its own XML Schema document, a tagged value “xsdDocument” shall be set providing a valid relative file name of the schema document.
  - The tagged value shall be set for every package representing the UML Application Schema.
  - All tagged values “xsdDocument” in a UML model shall be unique.
- Dependencies between packages shall be modelled explicitly.
  - Permission elements with stereotype «`import`» or unspecified dependency elements between packages shall be used to express the dependency of elements in a package from elements in another package.
  - All other dependency elements shall be ignored.

A.3. Classes (including Association Classes):

- All class names within the same Application Schema shall be unique and an “NCName” as defined by W3C XML Names.

- All classes shall have a stereotype specifying the meaning of the class.
  - Feature types shall be modelled as UML classes with stereotype «`FeatureType`».
  - Object types shall be modelled as UML classes with no stereotype. Object types are types where the instances shall have an identity, but which are not feature types.
  - UML classes with stereotype «`Type`» may have zero or more operations (these are not mapped to the GML application schema), attributes or associations.
  - The stereotype «`Abstract`» shall not be used in an Application Schema, because its use may be inconsistent with the use of correct UML notation, and thus misleading.
  - All instantiable subtypes of abstract types shall be either feature types, object types or data types.
  - Enumerations shall be modelled as UML classes with stereotype «`Enumeration`».
  - Code lists shall be modelled as UML classes with stereotype «`CodeList`».
  - Union types shall be modelled as UML classes with stereotype «`Union`».
  - All other data types shall be modelled as UML classes with stereotype «`DataType`».
  - Classes without a stereotype are treated as object types.
- UML classes of the ISO 19100 series of International Standards for which a GML base type has been specified in ISO 19136-1:2020 (see Table D.2 there) may be subclassed in the UML Application Schema.
  - In the subclasses, additional properties may be added or properties of the subtype may be redefined with a restricted multiplicity or domain of values.
- A generalization relationship may be specified only between two classes that are either:
  - both feature types,
  - both object types, or
  - both data types.
- If a class is a specialization of another class, then this class shall have only one supertype (no support for multiple inheritance).

A.4. Attributes

- Every UML attribute of an abstract type, feature type, object type, data type or union type shall have a name and a type.
    - The name shall be an “NCName” as defined by W3C XML Names.
    - If its multiplicity is not `1`, the multiplicity shall be specified explicitly.
    - An initial value may be specified for attributes with a number, string or enumeration type.
- The type shall either be a predefined type (basic types from ISO 19103:2015; see ISO 19136-1:2020, Table D.2) or a class defined in the UML model.
- Every UML attribute of an «`Enumeration`» class shall have a name.
    - The type information is left empty.
    - No multiplicity, ordering or initial value information shall be attached to the attribute.
- Every UML attribute of a «`CodeList`» class shall have a name.
    - The type information is left empty.
    - No multiplicity or ordering information shall be attached to the attribute.
    - An initial value may be specified to document a code for the code list value.
    - If no initial value is specified, the value (i.e., the attribute name) is used as the code.
- The properties of a UML class are not ordered.
    - To support the consistent ordering of the properties from the UML model in the conversion to XML Schema, a tagged value `sequenceNumber` (value domain: integer) shall be specified for every attribute.
    - The value shall be unique for all attributes and association ends of a class. Associations and association ends:
- Every UML association shall be an association with exactly two association ends.
    - Both association ends shall connect to a feature, object or data type and shall have no stereotype or the stereotype «`association`» (otherwise the whole association will be ignored).

- An association shall not contain any properties.
    - Association classes may contain properties.
- If an association end is navigable, it shall be marked as such and shall have a role name.
    - If a name is provided, it shall be an `NCName` as defined by W3C XML Names.
    - An association end with no name shall be ignored, even if it is marked as navigable.
    - The multiplicity shall be given explicitly.
    - The aggregation kind shall be specified explicitly if it is not `none`.
    - If the target class of an association end is a data type, then the aggregation kind shall be `composition`.

B

ANNEX B (INFORMATIVE) SHAPECHANGE CONFIGURATIONS

B

ANNEX B (INFORMATIVE) SHAPECHANGE CONFIGURATIONS

B.1. Introduction

ShapeChange provides a robust, complex, and sometimes intimidating, set of configuration options that enable extensive control over the processing of UML-based models. This Annex documents each ShapeChange configuration — an XML Instance document — that was employed in this task. These configurations may be used as starting points for exploring alternative model processing strategies specific to the target technology. Key decision points are documented as inline comments.

B.2. UML to GML

B.2.1. General

Developing a complete XML-based encoding of the CityGML 3.0 Conceptual Model requires determining encodings corresponding to UML classes defined in external conceptual models. When there exist authoritative encodings then UML classes in the external conceptual models can be mapped to individual corresponding encoding representations in XML Schema. When there do not exist authoritative encodings then it becomes necessary to develop those encodings in addition to developing the CityGML 3.0 Conceptual Model encoding itself.

As it happens, all external conceptual models upon which the CityGML 3.0 Conceptual Model is dependent have authoritative XML Schemas — published by either ISO or OASIS.

B.2.2. CityGML-derived XML Schema

This section provides a ShapeChange configuration used to convert the “CityGML Conceptual Model 3.0” models in Enterprise Architect EAP format into XSDs that conform to GML.

The GitHub ogc-mbs repository (https://github.com/metanorma/ogc-mbs/) integrates this configuration file in its continuous delivery workflow to generate XSDs from the EA XMI file `CityGML_3.0_Consolidated_Draft.xml` (exported from EA EAP).

ShapeChange (current development baseline, including the new XML Schema-related rule: `rule-xsd-cls-basictype-list`) was used to process the revised CityGML 3.0 Conceptual Model, as specified in a Sparx Systems Enterprise Architect Project (EAP) file.

The ShapeChange project configuration file used consists of two processing blocks, an `<input>` processing block loads the EA-based conceptual model into memory and then validates it with respect to a variety of UML and ISO 19100-series standards requirements (e.g., ISO 19103:2015 and ISO 19109:2015) following which a `<TargetXmlSchema>` processing block generates a set of XML Schema (`.xsd`) and Schematron (`.sch`) files corresponding to that in-memory model.

```
<?xml version="1.0" encoding="UTF-8"?>
<ShapeChangeConfiguration xmlns:xi="http://www.w3.org/2001/XInclude"
    xmlns="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
    xmlns:sc="http://www.interactive-instruments.de/ShapeChange/Configuration/
1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.interactive-instruments.de/ShapeChange/Confi
guration/1.1
http://shapechange.net/resources/schema/ShapeChangeConfiguration.xsd">

<input id="CityGML">
    <parameter name="inputModelType" value="EA7"/>
    <parameter name="inputFile" value="../CityGML_3.0_TB17_0930.eap"/>

    <parameter name="constraintLoading" value="enabled"/>
    <!-- Ignore constraints that are not specified using OCL -->
    <parameter name="oclConstraintTypeRegex" value="OCL"/>

    <xi:include href="http://shapechange.net/resources/config/StandardAliases.
xml"/>
    <stereotypeAliases>
      <!-- Establish CityGML-specific stereotypes as aliases for well-known
stereotypes. -->
      <StereotypeAlias wellknown="property" alias="Property"/>
      <StereotypeAlias wellknown="version" alias="Version"/>
      <!-- Stereotypes <<ObjectType>> and <<TopLevelFeatureType>> will be
ignored. -->
    </stereotypeAliases>

    <!-- Identify packages to be processed as Application Schemas. -->
    <parameter name="appSchemaNamespaceRegex" value="^http://www.opengis.net/
citygml/.*"/>
  </input>

  <log>
    <parameter name="reportLevel" value="INFO"/>
    <parameter name="logFile" value="./generateXSD_log.xml"/>
  </log>

  <targets>
    <TargetXmlSchema inputs=
"CityGML"
class="de.interactive_instruments.ShapeChange.Target.XmlSchema.
```

```
XmlSchema"
mode="enabled">
      <targetParameter name="outputDirectory" value="."/>

      <!-- Configure Schematron processing - file naming and XSLT Edition. -->
      <targetParameter name="schematronFileNameTemplate" value="[[SCHEMA_XSD_
BASENAME]].sch"/>
      <targetParameter name="segmentSchematron" value="true" />
      <targetParameter name="schematronQueryBinding" value="xslt2" />

      <!-- Include XML Annotation-based documentation -->
      <targetParameter name="includeDocumentation" value="true" />
      <targetParameter name="documentationTemplate" value="[[documentation]]" /
>
      <targetParameter name="documentationNoValue" value="[no value specified]"
 />

      <!-- Specify encoding rule to be used unless overridden by that assigned
to the UML element. -->
      <!--   Note that the CityGML 3.0 Conceptual Model inconsistently
populates TV xsdEncodingRule  -->
      <!--   with value "citygml". There are also two cases of "notEncoded".
                  -->
      <targetParameter name="defaultEncodingRule" value="citygml"/>

      <rules>
        <EncodingRule name="citygml" extends="iso19136_2007">
          <!-- Support the GML 3.3 association class encoding rule extension
for TextureAssociation. -->
          <rule name="rule-xsd-rel-association-classes"/>

          <!-- Basic types may be restricted with facets. For example, the
length of a subtype of CharacterString -->
          <!--   may be restricted through the use of the "length" tagged
value, or the allowed range of numeric  -->
          <!--   values may be limited through the use of the tagged values
"rangeMinimum" & "rangeMaximum".  -->
          <rule name="rule-xsd-cls-basictype"/>

          <!-- A basic type that matches this conversion rule and has a single
property with maximum multiplicity  -->
          <!--   greater than 1 will be converted as a list-based simple type.
The list item type is the XSD type  -->
          <!--   of the UML property value type. If the minimum multiplicity
of the UML property is 0 and the      -->
          <!--   maximum multiplicity is unbounded ('*'), then the length of
the resulting list is not restricted. -->
          <!--   Otherwise, length restrictions are defined according to the
multiplicity of the property.         -->
          <rule name="rule-xsd-cls-basictype-list"/>

          <!-- If a <<Union>> class has a tagged value "gmlAsGroup" with a
value of "true", then the class is    -->
          <!--   encoded as a global group which is referenced wherever a
property is defined that has the union -->
          <!--   class as its value. (Note that this is only valid if it is
clear from the context how to map    -->
          <!--   the individual values to the conceptual model.)
                                  -->
          <rule name="rule-xsd-cls-union-asGroup"/>

          <!-- If an attribute has an initial value, it is converted to a
default value in XML Schema.   -->
```

```
          <!--   If the attribute carries the constraint "{frozen}", too, the
initial value is converted -->
          <!--   to a fixed element value in XML Schema.
                        -->
          <rule name="rule-xsd-prop-initialValue"/>

          <!-- Generate constraining facets based on tagged values length,
maxLength, -->
          <!--    size, pattern, rangeMaximum, and rangeMinimum of a property.
       -->
          <rule name="rule-xsd-prop-constrainingFacets"/>

          <!-- Set default behavior to not generate inline annotation elements.
 -->
          <rule name="rule-xsd-all-no-documentation"/>

          <!-- Enables the OCL 2.2 parser and Schematron code-generator. -->
          <rule name="rule-xsd-pkg-schematron"/>
        </EncodingRule>
      </rules>

      <xi:include href="http://shapechange.net/resources/config/StandardNamespa
ces.xml"/>
      <xmlNamespaces>
        <!-- Establish XML namespace identifiers for all external schemas not
specified in StandardNamespaces.xml -->
       <XmlNamespace nsabr="xAL" ns="urn:oasis:names:tc:ciq:xal:
3"
location="http://docs.oasis-open.org/ciq/v3.0/cs02/xsd/default/xsd/xAL.xsd"/>

        <!-- Establish XML namespace identifiers for all CityGML schemas
             -->
        <!--   since these are not specified in the UML Model using package
tagged values. -->
       <XmlNamespace nsabr="app" ns="http://www.opengis.net/citygml/
appearance/3.0" location="./appearance.xsd"/>
       <XmlNamespace nsabr="brid" ns="http://www.opengis.net/citygml/bridge/3.
0" location="./bridge.xsd"/>
       <XmlNamespace nsabr="bldg" ns="http://www.opengis.net/citygml/
building/3.0" location="./building.xsd"/>
       <XmlNamespace nsabr="pcl" ns="http://www.opengis.net/citygml/
pointcloud/3.0" location="./pointCloud.xsd"/>
       <XmlNamespace nsabr="frn" ns="http://www.opengis.net/citygml/
cityfurniture/3.0" location="./cityFurniture.xsd"/>
       <XmlNamespace nsabr="grp" ns="http://www.opengis.net/citygml/
cityobjectgroup/3.0" location="./cityObjectGroup.xsd"/>
       <XmlNamespace nsabr="con" ns="http://www.opengis.net/citygml/
construction/3.0" location="./construction.xsd"/>
       <XmlNamespace nsabr="core" ns="http://www.opengis.net/citygml/3.0"
location="./cityGMLBase.xsd"/>
       <XmlNamespace nsabr="dyn" ns="http://www.opengis.net/citygml/
dynamizer/3.0" location="./dynamizer.xsd"/>
       <XmlNamespace nsabr="gen" ns="http://www.opengis.net/citygml/generics/3.
0" location="./generics.xsd"/>
       <XmlNamespace nsabr="luse" ns="http://www.opengis.net/citygml/landuse/3.
0" location="./landUse.xsd"/>
       <XmlNamespace nsabr="dem" ns="http://www.opengis.net/citygml/relief/3.0"
 location="./relief.xsd"/>
       <XmlNamespace nsabr="tran" ns="http://www.opengis.net/citygml/
transportation/3.0" location="./transportation.xsd"/>
       <XmlNamespace nsabr="tun" ns="http://www.opengis.net/citygml/tunnel/3.0"
 location="./tunnel.xsd"/>
```

```
        <XmlNamespace nsabr="veg" ns="http://www.opengis.net/citygml/
vegetation/3.0" location="./vegetation.xsd"/>
        <XmlNamespace nsabr="vers" ns="http://www.opengis.net/citygml/
versioning/3.0" location="./versioning.xsd"/>
        <XmlNamespace nsabr="wtr" ns="http://www.opengis.net/citygml/
waterbody/3.0" location="./waterBody.xsd"/>
      </xmlNamespaces>

      <!-- Establish XML Schema mappings from UML classes to XML elements,
types, attributes, or attributeGroups defined -->
      <!-- in external XML Schemas. In consequence suppresses generation of
class representation in the target schema.   -->
      <xi:include href="http://shapechange.net/resources/config/StandardMapEntr
ies.xml"/>

      <!-- Augment the existing set of XML Schema mappings specified in
StandardMapEntries.xml -->
      <xsdMapEntries>
        <!-- ISO 19103 -->
        <XsdMapEntry type="URI" xsdEncodingRules="citygml"
  xmlPropertyType="anyURI" xmlType="anyURI" xmlTypeType="simple"
  xmlTypeContent="simple"/>

        <!-- ISO 19109 -->
        <XsdMapEntry type="AnyFeature" xsdEncodingRules="citygml"
  xmlType="gml:AbstractFeatureType" xmlElement="gml:AbstractFeature"
  xmlPropertyType="gml:FeaturePropertyType"/>

        <!-- ISO 19111 -->
        <XsdMapEntry type="SC_EngineeringCRS" xsdEncodingRules="citygml"
  xmlType="gml:EngineeringCRSType" xmlElement="gml:EngineeringCRS"
  xmlPropertyType="_MP_"/>
        <XsdMapEntry type="SC_CRS" xsdEncodingRules="citygml"
  xmlAttributeGroup="gml:SRSReferenceGroup"/>

        <!-- ISO 19123 -->
        <XsdMapEntry type="CV_DiscreteGridPointCoverage" xsdEncodingRules=
"citygml"
  xmlType="gml:RectifiedGridCoverageType" xmlElement="gml:
RectifiedGridCoverage"
  xmlPropertyType="_P_"/>

        <!-- ISO 19136 -->
        <XsdMapEntry type="Code" xsdEncodingRules="citygml"
  xmlPropertyType="gml:CodeType" xmlType="gml:CodeType" xmlTypeType="complex"
  xmlTypeContent="simple"/>
        <XsdMapEntry type="doubleList" xsdEncodingRules="citygml"
  xmlPropertyType="gml:doubleList" xmlType="gml:doubleList" xmlTypeType=
"simple"
  xmlTypeContent="simple"/>
        <XsdMapEntry type="DoubleOrNilReasonList" xsdEncodingRules="citygml"
  xmlPropertyType="gml:DoubleOrNilReasonListType"
  xmlType="gml:DoubleOrNilReasonListType" xmlTypeType="complex"
  xmlTypeContent="simple"/>
        <XsdMapEntry type="MeasureOrNilReasonList" xsdEncodingRules="citygml"
  xmlPropertyType="gml:MeasureOrNilReasonListType"
  xmlType="gml:MeasureOrNilReasonListType" xmlTypeType="complex"
  xmlTypeContent="simple"/>

        <!-- OASIS xAL -->
        <XsdMapEntry type="XALAddress" xsdEncodingRules="citygml"
  xmlType="xAL:Address" xmlElement="xAL:Address" xmlPropertyType="_P_"/>
```

```
        <!-- Establish mappings from UML properties to XML elements defined in
external XML Schemas. -->
        <!--   In consequence suppresses generation of property representation
in the target schema. -->
        <XsdPropertyMapEntry property="AbstractFeature::featureID"/>
        <XsdPropertyMapEntry property="AbstractFeature::identifier"/>
        <XsdPropertyMapEntry property="AbstractFeature::name"/>
        <XsdPropertyMapEntry property="AbstractFeature::description"/>

        <XsdPropertyMapEntry property="ImplicitGeometry::objectID"/>
      </xsdMapEntries>
    </TargetXmlSchema>
  </targets>
</ShapeChangeConfiguration>
```

Figure B.1 — ShapeChange configuration to convert CityGML 3.0 EA UML into GML XSD files

In the course of loading and then processing the conceptual model, ShapeChange generates a log, documenting the execution of each stage of processing along with key processing findings categorized by their significance (e.g., Information, Warning, or Error). Log entries are further categorized as involving either process flow (`PF-` entries) or process-specific. See: https://shapechange.net/get-started/config/log/

A number of Warnings are logged while the `<input>` process validates the revised CityGML 3.0 Conceptual Model. There are no anomalous log entries stemming from execution of the `<TargetXmlSchema>` process.

```
PF-I ---------- Semantic validation of ShapeChange configuration: START -------
---
I Validating input parameters.
I Validation of input parameters completed.
I Validating log parameters.
I Validation of log parameters completed.
PF-I --- Validating target with @class 'de.interactive_instruments.ShapeChange.
Target.XmlSchema.XmlSchema' and @inputs 'CityGML' ...
PF-I ---------- Semantic validation of ShapeChange configuration: COMPLETE ----
------
I Connecting to D:\TB17 Model Driven Standards\SC_Work\makeXSDdirect\..
\CityGML_3.0_TB17_0712.eap
I Connected to D:\TB17 Model Driven Standards\SC_Work\makeXSDdirect\..\CityGML_
3.0_TB17_0712.eap
I Starting reading D:\TB17 Model Driven Standards\SC_Work\makeXSDdirect\..
\CityGML_3.0_TB17_0712.eap
I Finished reading D:\TB17 Model Driven Standards\SC_Work\makeXSDdirect\..
\CityGML_3.0_TB17_0712.eap
W The multiplicity value of 'size' is neither a number nor a known string. '*'
is used instead.
W ... Context: Class 'Model::ISO TC211::ISO 19103 All::ISO 19103:2005
Conceptual schema language::Basic Types::Primitive::Text::CharacterString'
W The multiplicity value of 'size' is neither a number nor a known string. '*'
is used instead.
W ... Context: Class 'Model::ISO TC211::ISO 19103 All::ISO 19103:2005
Conceptual schema language::Basic Types::Primitive::Text::CharacterString'
W Restriction of property 'boundary' in class 'HollowSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'WaterBody' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BridgeRoom' from supertype
'AbstractSpace'.
```

```
W Restriction of property 'boundary' in class 'BuildingRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'Storey' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstruction' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Door' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstructiveElement'
from supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractInstallation' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Window' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'TrafficSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AuxiliaryTrafficSpace' from
supertype 'AbstractSpace'.
PF-I Application schema found, package name: 'CityObjectGroup', target
namespace: 'http://www.opengis.net/citygml/cityobjectgroup/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Core', target namespace: 'http://
www.opengis.net/citygml/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'LandUse', target namespace:
'http://www.opengis.net/citygml/landuse/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Relief', target namespace: 'http:
//www.opengis.net/citygml/relief/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Tunnel', target namespace: 'http:
//www.opengis.net/citygml/tunnel/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Vegetation', target namespace:
'http://www.opengis.net/citygml/vegetation/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'WaterBody', target namespace:
'http://www.opengis.net/citygml/waterbody/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Appearance', target namespace:
'http://www.opengis.net/citygml/appearance/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Bridge', target namespace: 'http:
//www.opengis.net/citygml/bridge/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Building', target namespace:
'http://www.opengis.net/citygml/building/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Construction', target namespace:
'http://www.opengis.net/citygml/construction/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Versioning', target namespace:
'http://www.opengis.net/citygml/versioning/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Dynamizer', target namespace:
'http://www.opengis.net/citygml/dynamizer/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'PointCloud', target namespace:
'http://www.opengis.net/citygml/pointcloud/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'Generics', target namespace:
'http://www.opengis.net/citygml/generics/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
```

```
PF-I Application schema found, package name: 'Transportation', target
namespace: 'http://www.opengis.net/citygml/transportation/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Application schema found, package name: 'CityFurniture', target namespace:
 'http://www.opengis.net/citygml/cityfurniture/3.0'
PF-I Now processing target 'XML Schema' for input 'CityGML'.
PF-I Executed target class 'de.interactive_instruments.ShapeChange.Target.
XmlSchema.XmlSchema' for input ID: 'CityGML'.
-------------------------------------------------
```

Figure B.2 — ShapeChange log file for XML Schema

B.3. UML to JSON Schema

B.3.1. General

Developing a complete JSON-based encoding of the CityGML 3.0 Conceptual Model requires resolving encodings corresponding to references to types defined in external conceptual models. When there exist authoritative encodings then UML classes in the external conceptual models can be mapped to individual corresponding encoding representations in JSON Schema. When there do not exist authoritative encodings then it becomes necessary to develop those encodings in addition to developing the CityGML 3.0 Conceptual Model encoding itself.

As it happens, no external conceptual model upon which the CityGML 3.0 Conceptual Model is dependent has an authoritative JSON Schema. In consequence it becomes necessary to develop such JSON Schemas before the CityGML 3.0 Conceptual Model JSON Schema can be completed.

Unfortunately, it is the case that there exists no UML-based conceptual model for OASIS xAL and thus an MDA-based process cannot be applied; an OASIS xAL JSON Schema would need to be developed by other means (e.g., hand crafting).

B.3.2. ISO-derived JSON Schema

ShapeChange (current development baseline, including the new XML Schema-related rule: `rule-xsd-cls-basictype-list`) was used to process the revised ISO 19100-series Conceptual Models, as specified in a Sparx Systems Enterprise Architect Project (EAP) file.

The ShapeChange project configuration file used consists of three processing blocks, an `<input>` processing block loads the EA-based conceptual model into memory and then validates it with respect to a variety of UML and ISO 19100-series standards requirements (e.g., ISO 19103:2015 and ISO 19109:2015) following which a `<Transformer>` processing block revises the in-memory model, and then finally a `<TargetXmlSchema>` processing block generates a set of JSON Schema (`.json`) files corresponding to that revised in-memory model.

B.3.3. CityGML-derived JSON Schema

ShapeChange (current development baseline, including the new XML Schema-related rule: `rule-xsd-cls-basictype-list`) was used to process the revised CityGML 3.0 Conceptual Model, as specified in a Sparx Systems Enterprise Architect Project (EAP) file.

The ShapeChange project configuration file used consists of three processing blocks, an `<input>` processing block loads the EA-based conceptual model into memory and then validates it with respect to a variety of UML and ISO 19100-series standards requirements (e.g., ISO 19103:2015 and ISO 19109:2015) following which a `<Transformer>` processing block revises the in-memory model, and then finally a `<TargetXmlSchema>` processing block generates a set of JSON Schema (`.json`) files corresponding to that revised in-memory model.

```
<?xml version="1.0" encoding="UTF-8"?>
<ShapeChangeConfiguration xmlns:xi="http://www.w3.org/2001/XInclude"
    xmlns="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
    xmlns:sc="http://www.interactive-instruments.de/ShapeChange/Configuration/
1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.interactive-instruments.de/ShapeChange/Conf
iguration/1.1 http://shapechange.net/resources/schema/ShapeChangeConfiguration.
xsd">

  <input id="INPUT">
    <parameter name="inputModelType" value="EA7"/>
    <parameter name="inputFile" value="../CityGML_3.0_TB17_0930.eap"/>

    <xi:include href="http://shapechange.net/resources/config/StandardAliases.
xml"/>
    <stereotypeAliases>
      <!-- Establish CityGML-specific stereotypes as aliases for well-known
stereotypes. -->
      <StereotypeAlias wellknown="property" alias="Property"/>
      <StereotypeAlias wellknown="version" alias="Version"/>
      <StereotypeAlias wellknown="Type" alias="BasicType"/>
      <!-- Stereotypes <<ObjectType>> and <<TopLevelFeatureType>> will be
ignored. -->
    </stereotypeAliases>

    <!-- Identify packages to be processed as Application Schemas. -->
    <parameter name="appSchemaNamespaceRegex" value="^http://www.opengis.net/
citygml/.*"/>
  </input>

  <log>
    <parameter name="reportLevel" value="INFO"/>
    <parameter name="logFile" value="./generateJSON_log.xml"/>
  </log>

  <transformers>
    <!-- ************************************************************************
***** -->
    <!-- This transformation maps an association class into a nominally
equivalent   -->
```

```
    <!--   class plus two associations, as defined by the OGC GML 3.3 standard.
      -->
    <!-- See: https://shapechange.net/transformations/association-class-mapper
      -->
    <!-- ********************************************************************
***** -->
    <Transformer id="schema" input="INPUT"
 class="de.interactive_instruments.ShapeChange.Transformation.Flattening.
AssociationClassMapper" mode="enabled">
      <rules>
        <ProcessRuleSet name="mapper">
          <rule name="rule-trf-all-postprocess-skip-constraint-validation"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
  </transformers>


  <targets>
    <!-- Must be included first for its legacy XSD-related side-effects during
input processing. -->
    <TargetXmlSchema inputs="INPUT" class="de.interactive_instruments.
ShapeChange.Target.XmlSchema.XmlSchema" mode="enabled">
      <!-- Ignore target results -->
      <targetParameter name="outputDirectory" value="./ignore" />
      <targetParameter name="skipXmlSchemaOutput" value="true" />

      <!-- Specify encoding rule to be used unless overridden by that assigned
to the UML element. -->
      <!--   Note that the CityGML 3.0 Conceptual Model inconsistently
populates TV xsdEncodingRule  -->
      <!--   with value "citygml". There are also two cases of "notEncoded".
                  -->
      <targetParameter name="defaultEncodingRule" value="citygml" />
      <rules>
        <EncodingRule name="citygml" extends="iso19136_2007">
          <rule name="rule-xsd-cls-basictype"/>
          <rule name="rule-xsd-cls-basictype-list"/>
          <rule name="rule-xsd-cls-union-asGroup"/>
        </EncodingRule>
      </rules>

      <!-- No need to configure additional namespaces since the XSD output is
suppressed -->
      <xi:include href="http://shapechange.net/resources/config/StandardNamespa
ces.xml" />

      <!-- Must supply map entries despite the XSD output being suppressed -->
      <xi:include href="./CityGmlXsdMapEntries.xml" />
    </TargetXmlSchema>

    <Target inputs="schema" class="de.interactive_instruments.ShapeChange.
Target.JSON.JsonSchemaTarget" mode="enabled">
      <targetParameter name="outputDirectory" value="."/>
      <targetParameter name="sortedOutput" value="true"/>
      <targetParameter name="jsonSchemaVersion" value="2019-09"/>

      <!-- If the tagged value jsonBaseUri of an application schema has a non
empty value, then that value      -->
      <!--   will be used as base URI for all JSON Schemas produced for the
content of that application schema. -->
      <targetParameter name="jsonBaseUri" value="http://test.org/schema"/>
```

```
      <!-- The name of the JSON member to be added to a JSON object in order
to encode the type represented -->
      <!--   by that object. Only used with rule-json-cls-name-as-entityType.
                        -->
      <targetParameter name="entityTypeName" value="type"/>

      <!-- Defines the default value for tag inlineOrByReference of a UML
property,  -->
      <!--   in case that tag is undefined or has an empty value for the
property.   -->
<!--     <targetParameter name="inlineOrByReferenceDefault" value=
"inlineOrByReference"/> -->     <!-- NECESSARY? -->

      <!-- Reference to the JSON Schema definition which shall be added to a
feature type in order to represent  -->
      <!--   an additional generalization relationship. Only used with rule-
json-cls-virtualGeneralization.      -->
      <targetParameter name="baseJsonSchemaDefinitionForFeatureTypes" value=
"https://geojson.org/schema/Feature.json"/>
      <targetParameter name="baseJsonSchemaDefinitionForObjectTypes" value=
"https://geojson.org/schema/Feature.json"/>

      <!-- If set to “true”, a map entry file will be written for each
processed schema,        -->
      <!--   containing a map entry for each encoded type from that schema.
               -->
      <targetParameter name="writeMapEntries" value="true"/>


      <targetParameter name="defaultEncodingRule" value="cityJson"/>
      <rules>
        <EncodingRule name="cityJson" extends="defaultPlainJson">
          <!--  A geometry property is identified as default geometry by
setting tagged value   -->
          <!--    defaultGeometry on the property to the value true. That
property will then be -->
          <!--    encoded as a top-level "geometry" member. If multiple such
properties exist   -->
          <!--    (potentially inherited), none of them is selected as default
geometry         -->
          <!--    (because no informed choice can be made) and an error will
be logged.          -->
          <rule name="rule-json-cls-defaultGeometry-
multipleGeometryProperties"/>

          <!--  The identifier of a type with identity (feature type or object
type) will be    -->
          <!--    encoded using an identifier member that is provided by a
common base type      -->
          <!--    (e.g. the "id" member of a GeoJSON Feature, to which a
generalization          -->
          <!--    relationship exists for a given feature type. That means
that no additional   -->
          <!--    identifier property is created.
               -->
          <rule name="rule-json-cls-ignoreIdentifier"/>

          <!--  UML properties are encoded within the GeoJSON "properties"
member.              -->
          <!--    Notable exceptions are the GeoJSON members "id", "geometry",
and "bbox".      -->
          <rule name="rule-json-cls-nestedProperties"/>
```

```
          <!--  Add generalization relationships to specific kinds of classes
– if               -->
          <!--    a) according ShapeChange JSON Schema target parameters have
been set, and      -->
          <!--    b) the class does not already have that generalization
relationship via       -->
          <!--    one of its supertypes.
                  -->
          <rule name="rule-json-cls-virtualGeneralization"/>

          <!-- If a direct or indirect supertype of an application schema
class is mapped       -->
          <!--   to one of the simple JSON Schema types string, number,
integer, or boolean,    -->
          <!--   then it is treated as a "basic type" and mapped to a simple
JSON Schema type.  -->
          <!--   A basic type can be restricted using a number of JSON Schema
keywords.          -->
          <rule name="rule-json-cls-basictype"/>

          <!-- A union is converted to the JSON Schema definition of a JSON
object.              -->
          <!--   Each union option is represented as an optional member of the
JSON object.      -->
          <!--   The choice between the options defined by the union is
encoded using           -->
          <!--   "maxProperties" = "minProperties" = 1.
                  -->
          <rule name="rule-json-cls-union-propertyCount"/>

          <!-- All code lists are represented by a JSON Schema that restricts
the type to       -->
          <!--   "string", and states that the "format" is "uri".
                  -->
          <rule name="rule-json-cls-codelist-uri-format"/>

          <!-- Adds another JSON member to the JSON object which represents
the class that      -->
          <!--   is being converted. The JSON member is required and string-
valued. It should   -->
          <!--   be used to encode the name of the type that is represented by
the JSON object. -->
          <rule name="rule-json-cls-name-as-entityType"/>
        </EncodingRule>
      </rules>

      <xi:include href="./CityGmlJsonMapEntries.xml"/>
    </Target>
  </targets>
</ShapeChangeConfiguration>
```

Figure B.3 — ShapeChange project configuration file for CityGML-derived JSON Schema

In the course of loading and then processing the conceptual model ShapeChange generates a log, documenting the execution of each stage of processing along with key processing findings categorized by their significance (e.g., Information, Warning, or Error). Log entries are further categorized as involving either process flow (`PF-` entries) or process-specific. See: https://shapechange.net/get-started/config/log/

A number of Warnings are logged while the <input> process validates the revised CityGML 3.0 Conceptual Model. There are no anomalous log entries stemming from execution of the `<TargetXmlSchema>` process.

```
PF-I ---------- Semantic validation of ShapeChange configuration: START -------
---
I Validating input parameters.
I Validation of input parameters completed.
I Validating log parameters.
I Validation of log parameters completed.
PF-I --- Validating transformer with @id 'schema' ...
PF-I --- Validating transformer with @id 'TRF1' ...
PF-I --- Validating target with @class 'de.interactive_instruments.ShapeChange.
Target.XmlSchema.XmlSchema' and @inputs 'INPUT' ...
PF-I --- Validating target with @class 'de.interactive_instruments.ShapeChange.
Target.JSON.JsonSchemaTarget' and @inputs 'schema' ...
PF-I ---------- Semantic validation of ShapeChange configuration: COMPLETE ----
------
W Restriction of property 'boundary' in class 'HollowSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'WaterBody' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BridgeRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'BuildingRoom' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'Storey' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstruction' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Door' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractConstructiveElement'
from supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'AbstractInstallation' from
supertype 'AbstractSpace'.
W Restriction of property 'boundary' in class 'Window' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'TrafficSpace' from supertype
'AbstractSpace'.
W Restriction of property 'boundary' in class 'AuxiliaryTrafficSpace' from
supertype 'AbstractSpace'.
PF-I Application schema found, package name: 'CityObjectGroup', target
namespace: 'http://www.opengis.net/citygml/cityobjectgroup/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Core', target namespace: 'http://
www.opengis.net/citygml/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'LandUse', target namespace:
'http://www.opengis.net/citygml/landuse/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Relief', target namespace: 'http:
//www.opengis.net/citygml/relief/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Tunnel', target namespace: 'http:
//www.opengis.net/citygml/tunnel/3.0'
```

```
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Vegetation', target namespace:
'http://www.opengis.net/citygml/vegetation/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'WaterBody', target namespace:
'http://www.opengis.net/citygml/waterbody/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Appearance', target namespace:
'http://www.opengis.net/citygml/appearance/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Bridge', target namespace: 'http:
//www.opengis.net/citygml/bridge/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Building', target namespace:
'http://www.opengis.net/citygml/building/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Construction', target namespace:
'http://www.opengis.net/citygml/construction/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Versioning', target namespace:
'http://www.opengis.net/citygml/versioning/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Dynamizer', target namespace:
'http://www.opengis.net/citygml/dynamizer/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'PointCloud', target namespace:
'http://www.opengis.net/citygml/pointcloud/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Generics', target namespace:
'http://www.opengis.net/citygml/generics/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'Transportation', target
namespace: 'http://www.opengis.net/citygml/transportation/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Application schema found, package name: 'CityFurniture', target namespace:
 'http://www.opengis.net/citygml/cityfurniture/3.0'
PF-I Now processing target 'XML Schema' for input 'INPUT'.
I Skipping XML Schema output, as configured.
PF-I Executed target class 'de.interactive_instruments.ShapeChange.Target.
XmlSchema.XmlSchema' for input ID: 'INPUT'.
-------------------------------------------------
PF-I Now processing transformation 'schema' for input ID: 'INPUT'.
PF-I Performed transformation for transformer ID 'schema' for input ID:
'INPUT'.
-------------------------------------------------
PF-I Application schema found, package name: 'CityObjectGroup', target
namespace: 'http://www.opengis.net/citygml/cityobjectgroup/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Core', target namespace: 'http://
www.opengis.net/citygml/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
```

```
PF-I Application schema found, package name: 'LandUse', target namespace:
'http://www.opengis.net/citygml/landuse/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Relief', target namespace: 'http:
//www.opengis.net/citygml/relief/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Tunnel', target namespace: 'http:
//www.opengis.net/citygml/tunnel/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Vegetation', target namespace:
'http://www.opengis.net/citygml/vegetation/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'WaterBody', target namespace:
'http://www.opengis.net/citygml/waterbody/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Appearance', target namespace:
'http://www.opengis.net/citygml/appearance/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Bridge', target namespace: 'http:
//www.opengis.net/citygml/bridge/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Building', target namespace:
'http://www.opengis.net/citygml/building/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Construction', target namespace:
'http://www.opengis.net/citygml/construction/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Versioning', target namespace:
'http://www.opengis.net/citygml/versioning/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Dynamizer', target namespace:
'http://www.opengis.net/citygml/dynamizer/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'PointCloud', target namespace:
'http://www.opengis.net/citygml/pointcloud/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Generics', target namespace:
'http://www.opengis.net/citygml/generics/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'Transportation', target
namespace: 'http://www.opengis.net/citygml/transportation/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Application schema found, package name: 'CityFurniture', target namespace:
 'http://www.opengis.net/citygml/cityfurniture/3.0'
PF-I Now processing target 'JSON Schema' for input 'schema'.
PF-I Executed target class 'de.interactive_instruments.ShapeChange.Target.JSON.
JsonSchemaTarget' for input ID: 'schema'.
-------------------------------------------------
```

Figure B.4 — ShapeChange log file for CityGML-derived JSON Schema

C

ANNEX C (INFORMATIVE) BSI|RIBOSE SMART

C

ANNEX C (INFORMATIVE) BSI|RIBOSE SMART

C.1. General

BSI|Ribose SMART ("BR SMART") is a newly launched initiative that makes standards useable for native digital infrastructures.

Standards are increasingly used in situations that require machine interaction, such as in facilitating automated compliance. However, standards today are created in document form which cannot be directly used by machines.

BR SMART focuses on making standards immediately useable based on the vision of SMART (Standards that are Machine-Accessible, Readable and Transferrable) which is a newly proposed class of standards that aims to be digitally native. Led by several members of the group that conceived SMART, the program is jointly developed by the British Standards Institution (BSI) and Ribose.

Instead of taking a piecemeal approach that supplements published documents with digital hints, BR SMART revolutionizes the notion of standards by enriching content into digital native elements. In effect, BR SMART provides a business excellence framework powered by a full-stack suite of technologies and tools, covering standards usage from authoring, implementation, compliance to audits and assessments.

C.2. Summary

The concept of SMART was initially developed by the ISO Technical Management Board's Special Advisory Group on Machine-Readable Standards (ISO/TMBG/SAG_MRS) in 2019 to address user needs that require standards to be consumed by machines, tools, software, and humans via those channels. SMART has since been adopted in ISO's official "Strategy 2030" and a number of standardization organizations.

The SMART approach at ISO is now led by the ISO Council's SMART Steering Group ("ISO SMART SG") with its three sub-groups: the Business Model subgroup ("BM SG"), the Use Cases subgroup ("US SG") and the Technical Solutions subgroup ("TS SG").

The need for SMART is clear. Standard users today increasingly rely on technology, and often spend significant resources to manually transform standardized content from published documents into data that suits their systems. Manual conversion and implementation is error-prone and easily succumbs to interpretation ambiguity. SMART addresses this by representing a newly-defined class of standards that streamline content intention to standard users.

Existing approaches to the goals of SMART exist. As explained by ISO SAG_MRS in its final report, SMART standards have been offered in the domains of terminology standards, data modelling standards, registry standards, source code or schema standards, and requirement and testing standards, which consist of structured data that can easily be understood by machines.

The ISO TMB is currently running several SMART pilots in ISO/TC 46, ISO/TC 154, ISO/TC 184/SC 4 and ISO/TC 211 for such standards. The further challenge is on how to fully take advantage of the digital opportunity for standards that contain information that do not fit well as native digital content, while remaining compatible with the aforementioned structured data approach.

C.3. Approach

Instead of using text to represent standardized content, advances in technology now allow us to represent standards using fine-grained mechanisms in digital form, that enables native readability for systems and machines alike. While this sort of structuring can be done after the content is standardized, this work can be performed directly at the content creation stage for increased efficiency and improved effectiveness.

In BR SMART, we aim to help standards authors achieve the seemingly elusive “native SMART” standard — a standard authored by authors using a SMART-enabled workflow, which facilitates the end-to-end delivery of standardized content with integrity preserved — where the content intention is unambiguously conveyed to the standard user.

BR SMART provides end-to-end tooling for the development of these standards, from authoring, content delivery, to the usage of them. This approach provides opportunities for both open source and enterprise use cases.

C.4. “Function-first” standards

The major advantage of BR SMART is that its standards are functional. These “functional standards” provide machine-executable functionality that a standards user can directly utilize, such as to run simulations, execute algorithms, and allow adopting applications to store structured data according to schemas provided by the standard.

This function-first approach ensures that intended use cases are properly thought out at the onset of standards creation, and that a standard user can benefit from being able to use or simulate content specified in a standard, rather than having to read through the text and

imagining how things work. Just like the difference between test driving a vehicle on the road and being limited to reading vehicle specifications.

BR SMART defines the notion of “standards as models”. At its foundation, BR SMART is built on a new purpose-built, machine-executable modelling methodology allowing creation of unambiguous standards that can be directly used as user-facing functionality. This modelling technology is compatible with existing data model languages, such as UML and EXPRESS. The technology enables a BR SMART standard to be immediately useable, unambiguous, self-contained, modular, with elements that are atomic and reusable.

C.5. Developed and tested with a wide audience

The BR SMART initiative was developed in collaboration with a broad stakeholder base. That includes over 42 national standards committees, multiple standards development bodies, leading standards consultants and an expert pool of standards users.

Furthermore, the BR SMART approach has been validated by multiple successful pilot projects across various industries ranging from smart manufacturing, to information processing and management systems.

C.6. Legacy friendly

The BR SMART approach is legacy-friendly — it allows generating document formats and other types of outputs typically used by standards publishers, such as in XML, HTML and PDF formats. While the functions of these outputs are limited compared to the usage in the native BR SMART format, they present an excellent alternative to standards users who require these traditional representation formats.

C.7. Potential integration with existing Model-Based standards

The OGC Compliance Interoperability & Testing Evaluation (CITE) environment provides an automated testing framework that allows machine-driven compliance testing against systems that offer APIs specified by OGC Standards. CITE is used in the objective assessment required for the OGC certification programs.

While CITE tests of OGC Standards are realized in executable code, the tests originate as specifications from OGC Standards, and require a manual conversion and interpretation step before being able to be useable within CITE for automated testing.

The approach of BR SMART can potentially serve as the “processing bridge” that links up compliance tests from within OGC Standards to the automated format in CITE, facilitating a seamless conversion between documented tests and their automated counterparts. This potentially frees up resources and expertise for the standards publisher and also facilitates a tighter feedback look between the standards editors and the automated tests.

D

ANNEX D (INFORMATIVE) CLARIFICATIONS TO MODSPEC

ANNEX D (INFORMATIVE) CLARIFICATIONS TO MODSPEC

D.1. ModSpec issue 1: steps vs parts

We present a sample requirement and its corresponding conformance test from the CityGML standard below.

Table D.1 — Sample requirement

REQUIREMENT 18	/REQ/GENERICS/ADE/USE
ADE element and property extensions SHALL NOT be used unless conformance with the ADE Requirements Class can be demonstrated.	

Table D.2 — Corresponding test for the sample requirement

ABSTRACT TEST 18	/ATS/GENERICS/ADE/USE
Test Purpose	To validate that Application Data Extensions are not used unless conformance with the ADE Requirements Class can be demonstrated.
Requirement	/req/generics/ade_use
Test Method	Manual Inspection
If any ADE classes or properties are included in the Generics Package:	
A	Validate that the Implementation Specification conforms with the ADE Requirements Class

However, the “Conformance test” definition according to ModSpec notes OGC 08-131r3, Clause C.11:

> The requirements in a specification have to be tested and the conformance test specification contains the test’s definition. In this standard, conditional tests,

based on a requirement with some precondition, have as part of their execution the task of creating the precondition in which they will be tested. If the condition is not creatable, then the test is not required to be executed.

```
Enumeration TestType
{
   basic
   capabilities
}

Class ConformanceTest
{
  testPurpose: String
  testMethod: String
  Reference: String
  testType: TestType
  requirement: Requirement[1...*]
}
```

While the pseudocode definition in C.11 did not allow for test steps, it is clear that the definition text intended to support preconditions and test steps.

Considering the intention of the ModSpec, a reading of the ModSpec indicates that both “parts” and “sequential steps” are allowable:

- supporting parts: OGC 08-131r3, Clause 4.3 says “conformance test case: test for a particular requirement or a set of related requirements. NOTE When no ambiguity, the word — “case” may be omitted. i.e. conformance test is the same as conformance.`”
    - This NOTE implies that a “conformance test” may or may not be a test case, i.e. a “conformance test” may contain multiple “conformance test cases”.
- supporting sequential steps: OGC 08-131r3, Clause C.11 says “In this standard, conditional tests, based on a requirement with some precondition, have as part of their execution the task of creating the precondition in which they will be tested”.
    - This implies that a test case can contain multiple steps.
- supporting none.
    - OGC 08-131r3, Clause C.2 provides a UML diagram which does not provide for parts or steps, and the OGC 08-131r3, Clause C.11 UML definition supports neither too.

Given the above information we can safely conclude that the ModSpec standard should be updated to fully reflect its intention.

In order to accommodate existing OGC ModSpec instances, in this Testbed and in the OGC Metanorma implementation, we have updated the ModSpec implementation models to reflect the missing “test parts” and “sequential steps” concepts, as shown in Figure D.1:

```
// A Conformance Test includes potentially multiple Test Methods
// A Conformance Test Method is considered a "part" of the test (A, B...)
```

```
class ConformanceTest {
  testPurpose: String
  testType: TestType
  reference: RichText[0..*]
  linkToSpecification: RequirementOrRequirementPart[1..*]
  testMethod: ConformanceTestMethod[0..*]
}

// A Test Method can contain multiple steps
// A Test Method can be linked to a particular Requirement or Requirement Part
class ConformanceTestMethod {
  linkToSpecification: RequirementOrRequirementPart[0..*]
  steps: ConformanceTestStep[1..*]
}

// A Test Step can contain inner Test Steps
class ConformanceTestStep {
  content: RichText[1..*]
  inner: ConformanceTestStep[0..*]
}
```

Figure D.1 — ModSpec implementation of "test parts" and "sequential steps"

D.2. ModSpec issue 2: conformance tests with conditions

> *A Conformance class may be parameterized*. This means that the class' tests depend on some parameter that must be defined before the tests can be executed. For example, if a XYZ conformance class needs to specify a data format such as GML or KML to be tested, then XYZ(GML) is XYZ using GML, and XYZ(KML) is XYZ using KML. *Because the parameters choose which requirements will be tested, two conformance classes with distinct parameters should be considered as distinct conformance classes*.

(italicized text originally in red italics)

ModSpec does not allow conditions in tests. However, the abstract test in Table D.3 provides a statement that conflicts with this definition.

Table D.3 — Abstract test with condition

ABSTRACT TEST 18	/ATS/GENERICS/ADE/USE
Test Purpose	To validate that Application Data Extensions are not used unless conformance with the ADE Requirements Class can be demonstrated.
Requirement	/req/generics/ade_use
Test Method	Manual Inspection
If any ADE classes or properties are included in the Generics Package:	

ABSTRACT TEST 18	/ATS/GENERICS/ADE/USE
A	Validate that the Implementation Specification conforms with the ADE Requirements Class

The line "*If any ADE classes or properties are included in the Generics Package:*" is "like" a condition or parameter. We have seen "If…" and "For all…" patterns in abstract tests.

However, ModSpec itself indicates that this is an incompatible use:

1. OGC 08-131r3, Clause 6.5.1 says only conformance classes can be parameterized, not conformance tests.
2. OGC 08-131r3, Clause 6.5.1 also says that parameterization requires the conformance classes to be distinct, i.e. `XYZ(GML)` and `XYZ(KML)` are separate conformance classes.
3. Recall that OGC 08-131r3, Clause C.11 "Conformance tests" also specifies the following:
   - "In this standard, conditional tests, based on a requirement with some precondition, have as part of their execution the task of creating the precondition in which they will be tested".

The above information implies that a conformance test can include preconditions, as long as the precondition is created as a step within the conformance test. We can conclude that this is more of a matter of "how" to fit these abstract tests into ModSpec models.

Several options were presented to the Testbed in order to fit this information into ModSpec models:

1. Apply the same parameterization principles to Conformance Tests in addition to Conformance Classes.
2. Treat the quoted line ("If …") not as a condition or parameter, but internalized by individual parts, i.e.
   - Abstract Test 18A would be "If any ADE classes …, validate that the Implementation Specification…"
   - Abstract Test 18B would be "If any ADE classes …, {original B content}…", etc.

3. Treat the quoted line ("If …") as a "precondition" — the first step in a sequence of steps as specified in C.11 — i.e.

    - Step 1 "`Check if there are any ADE classes …, if none, return."
    - Step 2 "Validate that all classes from Step 1 conform with the Implementation Specification…" (content from 18A)
    - Step 3 "Validate that all classes from Step 1 conform with …" etc. (content from 18B)

4. Allow hierarchical steps in a Conformance Test Case, and do 2 but in a hierarchical manner (i.e. "If …, do {step A}, {step B} and {step C}").

Testbed participants have concluded that List 2 1) and List 2 4) are the selected methods to proceed.

The ModSpec implementation within Metanorma has been updated to reflect this latest understanding.

D.3. ModSpec issue 3: terminology and representation in OGC documents

Terminology of ModSpec objects differs from its original specification and in OGC documents.

Specifically in ModSpec:

- the Conformance Test is commonly called "Abstract Test" in OGC documents;
- the Conformance Test Suite is commonly called "Abstract Test Suite" in OGC documents.

In ModSpec, the nature of "abstract" is an attribute of the conformance test, not a type of conformance test.

Upon consultation with the OGC DocTeam and Testbed participants, it was decided that:

- The "Conformance Test" term should remain as "Conformance Test" instead of "Abstract Test".
- The term "Abstract Test Suite" is to remain as a mandatory section of OGC Standards as this usage has been widely adopted in OGC.

In addition, the representation of ModSpec models is inconsistent across OGC documents.

- In the DGGS document, the “Conformance Class” is represented as a ModSpec model, complete with the name, URI, and contents.
- In the CityGML 3.0 document, the “Conformance Class” is represented as an annex section, with no URI or description.

The recommendation is to use the ModSpec model representation for “Conformance Class” consistently across OGC.

In OGC documents, not all requirements and conformance tests cross-link with each other. This presents a difficulty in navigation: when viewing a requirement and if one wants to see the associated conformance test, the reverse operation is not possible.

It was agreed with the OGC DocTeam and Testbed participants that all corresponding ModSpec models, such as the “Requirement” and “Conformance Test”, “Requirements Class” and “Conformance Class”, should have cross-references to each other to support easy navigation.

D.4. ModSpec issue 4: Mismatch of UML diagrams, UML model definitions and textual description

As described in issues above, some of the UML diagrams, UML model definitions and textual descriptions offer conflicting information.

Given that ModSpec is used across all OGC Standards, it is important for the ModSpec specification to be clear and unambiguous.

It is recommended to review and update the OGC ModSpec to address these inconsistencies.

E

ANNEX E (INFORMATIVE) MODSPEC ENCODING SYNTAX IN METANORMA

ANNEX E (INFORMATIVE) MODSPEC ENCODING SYNTAX IN METANORMA

E.1. General

A requirement (in its generic meaning) is encoded via tagged example blocks containing other tagged example blocks and open blocks.

There are two ways to encode a ModSpec model:

1. Via definition lists
2. Via block attributes

The definition lists method is generally recommended for its multi-line syntax but some authors may prefer specifying attributes in the header.

NOTE: These two methods originate from Metanorma's general support of the alternative syntaxes for specifying a block attribute list as a definition list. The extension of the definition lists syntax to components is specific to Metanorma OGC.

E.2. Specifying using definition lists or block attributes

In Metanorma, the following two encoding syntaxes are considered equivalent.

- In the **block attributes** syntax, the necessary information is provided as an attribute list to the block. Values contained in the attribute list must be in plain text.
- In the **definition list** syntax, a `[%metadata]` definition list within a ModSpec model provides the necessary information for the specified model. Values given in the definition list syntax can be fully-formatted Metanorma AsciiDoc text.

Attributes that can take rich textual input (Metanorma AsciiDoc input), such as `part` and `conditions`, are components of requirements in Metanorma: these are encoded in the block attributes syntax using the `[.component]` role within the ModSpec model block, on open blocks or example blocks.

EXAMPLE:

```
[.component,class=part]
--
Part A of the requirement.
--

[.component,class=part]
====
Part A of the requirement.
====
```

Conversely, in definition list syntax, not only components such as `part` and `conditions`, but also `description` for descriptive text, can be specified in the definition list. (In block attributes syntax, descriptive text is left as normal text.)

The definition list may contain embedded levels [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.3]; this is needed specifically for steps embedded within a test method.

If you need to insert a cross-reference to a component, for example referencing a specific part of a requirement elsewhere, you can only use the block attributes sequence (as illustrated above).

The following two examples, demonstrating encoding of a ModSpec requirement, are encoded in Metanorma XML identically (and therefore rendered identically in output).

```
[requirement]
.Encoding of logical models
====
[%metadata]
label:: ogc/spec/waterml/2.0/req/xsd-xml-rules
subject:: system
part:: Metadata models faithful to the original UML model.
description:: Logical models encoded as XSDs should be faithful to the original
UML conceptual models.

test-method::
step::: Step 1
step::: Step 2
step:::: Step 2a
step:::: Step 2b
step::: Step 3
====
```

Figure E.1 — ModSpec requirement in definition list syntax

```
[requirement,label="ogc/spec/waterml/2.0/req/xsd-xml-rules",subject="system"]
.Encoding of logical models
====

[.component,class=part]
```

```
--
Metadata models faithful to the original UML model.
--

[.component,class=test-method]
-----
[.component,class=step]
------
Step 1
------

[.component,class=step]
------
Step 2

[.component,class=step]
-------
Step 2a
-------

[.component,class=step]
-------
Step 2b
-------
------

[.component,class=step]
------
Step 3
------
-----

Logical models encoded as XSDs should be faithful to the original UML
conceptual
models.
====
```

Figure E.2 — ModSpec requirement in block attributes syntax

These two syntaxes can be mixed.

E.3. ModSpec model attributes

A ModSpec model is encoded with one of these block types:

- `[requirement]` for Requirement
- `[recommendation]` for Recommendation
- `[permission]` for Permission
- `[requirements_class]` for Requirements class
- `[conformance_test]` for Conformance test

- `[conformance_class]` for Conformance class
- `[abstract_test]` for Abstract test

NOTE:These ModSpec types are available as of https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.3

In addition, if the Metanorma generic `[requirements]` block is used, these values are to be used in the `type` attribute.

The following two encodings are equivalent:

```
[conformance_test]
[requirement,type=conformance_test]
```

The following are additional attributes of the requirement block:

- `label` (mandatory). Label of the model, typically a URI. This must be unique in the document (as required by ModSpec), and is also used for referencing. Plain text.
- `subject` (optional). Subject that the model refers to. Plain text.
- `obligation` (optional). One of:
  - `requirement` (default)
  - `recommendation`
  - `permission`

Differentiated types of ModSpec models allow additional attributes.

E.4. Requirement, recommendation, permission

A Requirement (or Recommendation, Permission) is encoded as a `requirement`, `recommendation`, or `permission` block or by setting `type` to `requirement`, `recommendation`, or `permission`.

It supports the following attributes in addition to base ModSpec attributes:

- `conditions` (optional). Conditions on where this requirement applies. Accepts rich text.
- `part` (optional). A requirement can contain multiple parts of sub-requirements. Accepts rich text.
- `inherit` (optional). A requirement can inherit from one or more requirements. Accepts cross-references to other requirement models, or plain text. In block attributes syntax, accepts multiple semicolon-delimited values, which each could be a cross-reference to another conformance class or plain text. Can be repeated in definition list syntax.

- `classification` (optional). Classification of this requirement. The `classification` attribute is marked up as in the rest of Metanorma: `key1=value1;key2=value2...`, where *value* is either a single string, or a comma-delimited list of values.

NOTE:Support for `conditions`, `part` was added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.2.

EXAMPLE:

```
[requirement,label="/req/relief/classes"]
====
For each UML class defined or referenced in the Relief Package:

[.component,class=part]
--
The Implementation Specification SHALL contain an element which represents the
same concept as that defined for the UML class.
--

[.component,class=part]
--
The Implementation Specification SHALL represent associations with the same
source, target, direction, roles, and multiplicities as those of the UML class.
--
====
```

Figure E.3 — OGC CityGML 3.0 sample requirement with two parts (block attributes)

```
[requirement]
====
[%metadata]
label:: /req/relief/classes
description:: For each UML class defined or referenced in the Relief Package:
part:: The Implementation Specification SHALL contain an element which
represents the
same concept as that defined for the UML class.
part:: The Implementation Specification SHALL represent associations with the
same
source, target, direction, roles, and multiplicities as those of the UML class.
====
```

Figure E.4 — OGC CityGML 3.0 sample requirement with two parts (definition list)

```
[requirement,id="/req/core/encoding",label="/req/core/encoding"]
====
All target implementations SHALL conform to the appropriate GroundWaterML2
Logical Model UML defined in Section 8.
====
```

Figure E.5 — OGC GroundWaterML 2.0 sample requirement

E.5. Requirements class

A “Requirements class” is encoded as a block of `requirements_class` or using `type` equals to `requirements_class`.

A Requirements class is cross-referenced and captioned as a “{Requirement} class {N}” [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v0.2.11].

NOTE 1: Classes for Recommendations will be captioned as “Recommendations class {N}”, similarly for “Requirements class {N}” and “Permissions class {N}”.

Requirements classes allow the following attributes in addition to the base ModSpec attributes:

- Name (mandatory). The name of the requirements class should be specified as the block caption.
- `subject` (mandatory). The Target Type. Rendered as *Target Type*.
- `inherit` (optional). Dependent requirements classes. See Annex E.4.
- Embedded requirements (optional). Requirements contained in a class are marked up as nested requirements.

```
[requirements_class]
====
[%metadata]
label:: http://www.opengis.net/spec/CityGML-1/3.0/req/req-class-building
subject:: Implementation Specification
inherit:: <<rc_core,/req/req-class-core>>
inherit:: <<rc_construction,/req/req-class-construction>>
====
```

Figure E.6 — Example from OGC CityGML 3.0

NOTE 2: In this example, both block attributes and definition list syntax is used; the `inherit` attribute has two hyperlink values, which are expressed in the definition list.

A requirements class can contain multiple requirements, specified with embedded requirements.

The contents of these embedded requirements may be specified within the requirements class, or specified outside of the requirements class (referenced using the label). If the requirement is specified within a definition list, the definition list value is interpreted as the requirement label.

```
[requirements_class,inherit="urn:iso:dis:iso:19156:clause:7.2.2;urn:iso:
dis:iso:19156:clause:8;http://www.opengis.net/doc/IS/GML/3.2/clause/2.4;O&M
Abstract model, OGC 10-004r3, clause D.3.4;http://www.opengis.net/spec/SWE/2.0/
req/core/core-concepts-used"]
.GWML2 core logical model
====
```

```
[%metadata]
subject:: Encoding of logical models
label:: http://www.opengis.net/spec/waterml/2.0/req/xsd-xml-rules[*req/core*]
[requirement,label="/req/core/encoding"]
======
======

[requirement,label="/req/core/quantities-uom"]
======
======
====
```

Figure E.7 — Example from OGC GroundWaterML 2.0 (block attributes)

```
[requirements_class]
.GWML2 core logical model
====
[%metadata]
label:: http://www.opengis.net/spec/waterml/2.0/req/xsd-xml-rules[*req/core*]
obligation:: requirement
subject:: Encoding of logical models
inherit:: urn:iso:dis:iso:19156:clause:7.2.2
inherit:: urn:iso:dis:iso:19156:clause:8
inherit:: http://www.opengis.net/doc/IS/GML/3.2/clause/2.4
inherit:: O&M Abstract model, OGC 10-004r3, clause D.3.4
inherit:: http://www.opengis.net/spec/SWE/2.0/req/core/core-concepts-used
requirement:: /req/core/encoding
requirement:: /req/quantities-uom
====
```

Figure E.8 — Example from OGC GroundWaterML 2.0 (definition list)

renders as:

Requirement Class 1
GWML2 core logical model

req/core	
Obligation	Requirement
Target Type	Encoding of logical models
Dependency	urn:iso:dis:iso:19156:clause:7.2.2
Dependency	urn:iso:dis:iso:19156:clause:8
Dependency	http://www.opengis.net/doc/IS/GML/3.2/clause/2.4
Dependency	O&M Abstract model, OGC 10-004r3, clause D.3.4
Dependency	http://www.opengis.net/spec/SWE/2.0/req/core/core-concepts-used

Requirement	/req/core/encoding
Requirement	/req/core/quantities-uom

Embedded requirements (such as are found within Requirements classes) will automatically insert cross-references to the non-embedded requirements with the same label [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.0.8]:

```
[requirements_class,label="/req/conceptual"]
.GWML2 core logical model
====

[requirement,label="/req/core/encoding"]
======
======

====

[requirement,label="/req/core/encoding"]
====
Encoding requirement
====
```

renders as:

Requirement Class 3: GWML2 core logical model
/req/conceptual

Requirement 1	/req/core/encoding

Requirement 1 /req/core/encoding

Encoding requirement

E.6. Conformance class

Specified by setting the block as `conformance_class` or by using `type` as `conformance_class`.

A Conformance class is cross-referenced and captioned as “Conformance class {N}”, and is otherwise rendered identically to a “Requirements class” [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.0.4].

Conformance classes support the following attributes in addition to base ModSpec attributes:

- `subject`. Associated Requirements class. May be encoded as a cross-reference or as plain text. Rendered as *Requirements Class*.
- `inherit` (optional). Dependencies of the conformance class. Accepts multiple values, which each could be a cross-reference to another conformance class or plain text. See Annex E.4.

Conformance classes also feature:

- Name (optional). Specified as the block caption.
- Nesting (optional). Conformance tests contained in a conformance class are encoded as conformance tests within the conformance class block. See Requirements class.

NOTE:Conformance classes do not have a Target Type (as specified in ModSpec). If one must be encoded, it should be encoded as a classification key-value pair.

EXAMPLE:

```
[conformance_class,label="http://www.opengis.net/spec/ogcapi-features-2/1.0/
conf/crs",inherit="http://www.opengis.net/doc/IS/ogcapi-features-1/1.0#ats_
core",classification="Target Type:Web API"]
====
[%metadata]
subject:: <<rc_crs,Requirements Class 'Coordinate Reference Systems by
Reference'>>
====

[conformance_class]
====
[%metadata]
label:: http://www.opengis.net/spec/ogcapi-features-2/1.0/conf/crs
subject:: <<rc_crs,Requirements Class 'Coordinate Reference Systems by
Reference'>>
inherit:: http://www.opengis.net/doc/IS/ogcapi-features-1/1.0#ats_core
classification:: Target Type:Web API
====
```

renders as:

CONFORMANCE CLASS 1

http://www.opengis.net/spec/ogcapi-features-2/1.0/conf/crs

Requirements Class	*Requirements Class 'Coordinate Reference Systems by Reference'*
Dependency	http://www.opengis.net/doc/IS/ogcapi-features-1/1.0#ats_core
Target Type	Web API

E.7. Conformance test and Abstract test

A “Conformance test” can be “concrete” or “abstract” depending on the type of conformance test suite (see OGC 08-131r3, 6.4).

The OGC author should identify whether a standard requires an “Abstract test suite” or a “Conformance test suite” in order to decide the encoding of “Conformance tests” (concrete tests) versus “Abstract tests”.

- A conformance test is specified by creating a `conformance_test` block or using `type` as `conformance_test`. It is cross-referenced as “Conformance test {N}”
- An abstract test is specified by creating an `abstract_test` block or using `type` as `abstract_test`, or `conformance_test` together with `abstract=true`. It is cross-referenced as “Abstract test {N}” [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.0.4].

Conformance tests support the following attributes and components in addition to base ModSpec attributes:

- `subject`. The associated requirement. May be encoded as a cross-reference or plain text. Multiple semicolon-delimited values may be provided. Rendered as *Requirement*.
- `inherit` (optional). Dependencies. Accepts multiple values. Each may be a cross-reference or in plain text. See Annex E.4.
- Components (optional). Components of the conformance test. Accepts rich text. [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.0]. Allows the following classes:
  - `test-purpose` (optional). Purpose of the test. Rich text. Presented as *Test Purpose* [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.2]
  - `test-method` (optional). Method of the test. Rich text. Presented as *Test Method* [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.2]
  - `step` (optional). Step of the test method. Is expected to be embedded within `test-method`, and may contain substeps of its own. Rich text. Presented as a numbered list. added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.2]
  - `test-method-type` (optional). Method of the test. Rich text. Presented as *Test Method Type* [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.3]
  - `reference` (optional). Purpose of the test. Rich text. Presented as *Reference*.
- Test type of a Conformance test is encoded as a `classification` key-value pair.

Conformance tests also feature:

- Name (optional). Specified as the requirement's block caption.

NOTE:Conformance Tests are excluded from the "Table of Requirements" in Word output [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v0.2.10].

EXAMPLE:

```
[abstract_test,label="/ats/core/classes"]
====
[%metadata]
subject:: <<req_core_classes,/req/core/classes>>
[.component,class=test-purpose]
--
To validate that the Implementation Specification correctly implements the UML
Classes defined in the Conceptual Model.
--

[.component,class=test-method-type]
--
Manual Inspection
--

For each UML class defined or referenced in the Core Package:

[.component,class=part]
--
Validate that the Implementation Specification contains a data element which
represents the same concept as that defined for the UML class.
--

[.component,class=part]
--
Validate that the data element has the same relationships with other elements
as
those defined for the UML class. Validate that those relationships have the
same
source, target, direction, roles, and multiplicities as those documented in the
Conceptual Model.
--
====
```

Figure E.9 — Example of Abstract test from CityGML 3.0 (block attributes)

```
[abstract_test]
====
[%metadata]
label:: /ats/core/classes

subject:: <<req_core_classes,/req/core/classes>>
test-purpose:: To validate that the Implementation Specification correctly
implements the UML Classes defined in the Conceptual Model.

test-method-type:: Manual Inspection

description:: For each UML class defined or referenced in the Core Package:

part:: Validate that the Implementation Specification contains a data element
which represents the same concept as that defined for the UML class.
```

```
part:: Validate that the data element has the same relationships with other
elements as those defined for the UML class. Validate that those relationships
have the same source, target, direction, roles, and multiplicities as those
documented in the Conceptual Model.
====
```

Figure E.10 — Example of Abstract test from CityGML 3.0 (definition list)

```
[abstract_test,label="/conf/crs/crs-uri",classification="Test Type:Basic"]
====
[%metadata]
subject:: <<req_crs_crs-uri,/req/crs/crs-uri>>
subject:: <<req_crs_fc-md-crs-list_A,/req/crs/fc-md-crs-list A>>
subject:: <<req_crs_fc-md-storageCrs,/req/crs/fc-md-storageCrs>>
subject:: <<req_crs_fc-md-crs-list-global,/req/crs/fc-md-crs-list-global>>"

[.component,class=test-purpose]
--
Verify that each CRS identifier is a valid value
--

[.component,class=test-method]
--
For each string value in a `crs` or `storageCrs` property in the collections
and collection objects,
validate that the string conforms to the generic URI syntax as specified by
https://tools.ietf.org/html/rfc3986#section-3[RFC 3986, section 3].

. For http-URIs (starting with `http:`) validate that the string conforms to
the syntax specified by RFC 7230, section 2.7.1.

. For https-URIs (starting with `https:`) validate that the string conforms to
the syntax specified by RFC 7230, section 2.7.2.
--

[.component,class=reference]
--
<<ogc_07_147r2,clause=15.2.2>>
--

====
```

Figure E.11 — Example of Abstract test from DGGS (block attributes)

```
[abstract_test]
====
[%metadata]
label:: /conf/crs/crs-uri
subject:: <<req_crs_crs-uri,/req/crs/crs-uri>>
subject:: <<req_crs_fc-md-crs-list_A,/req/crs/fc-md-crs-list A>>
subject:: <<req_crs_fc-md-storageCrs,/req/crs/fc-md-storageCrs>>
subject:: <<req_crs_fc-md-crs-list-global,/req/crs/fc-md-crs-list-global>>"
classification:: Test Type:Basic
test-purpose:: Verify that each CRS identifier is a valid value
test-method::
+
--
```

```
For each string value in a `crs` or `storageCrs` property in the collections
and collection objects,
validate that the string conforms to the generic URI syntax as specified by
https://tools.ietf.org/html/rfc3986#section-3[RFC 3986, section 3].

. For http-URIs (starting with `http:`) validate that the string conforms to
the syntax specified by RFC 7230, section 2.7.1.

. For https-URIs (starting with `https:`) validate that the string conforms to
the syntax specified by RFC 7230, section 2.7.2.
--
reference:: <<ogc_07_147r2,clause=15.2.2>>
====
```

Figure E.12 — Example of Abstract test from DGGS (definitions list)

renders as:

ABSTRACT TEST 1	
/conf/crs/crs-uri	
Requirement	*/req/crs/crs-uri*, */req/crs/fc-md-crs-list A*, */req/crs/fc-md-storageCrs*, */req/crs/fc-md-crs-list-global*
Test Purpose	Verify that each CRS identifier is a valid value
Test Method	For each string value in a `crs` or `storageCrs` property in the collections and collection objects, validate that the string conforms to the generic URI syntax as specified by RFC 3986, section 3. 1. For http-URIs (starting with `http:`) validate that the string conforms to the syntax specified by RFC 7230, section 2.7.1. 2. For https-URIs (starting with `https:`) validate that the string conforms to the syntax specified by RFC 7230, section 2.7.2.
Reference	OGC-07-147r2: cl. 15.2.2
Test Type	Basic

E.8. Rendering of ModSpec models

OGC ModSpec models are rendered as tables.

NOTE: This rendering method is consistent with prior OGC practice.

- For HTML rendering, the CSS class of the ModSpec specification table is the `type` attribute of the requirement.

  The following types are recognized:

  - No value for Requirements
  - `conformance_test` for Conformance tests
  - `abstract_test` for Abstract tests
  - `requirements_class` for Requirements classes
  - `conformance_class` for Conformance classes

  The default CSS class currently assigned for HTML rendering is `recommend`.

- The heading of the table (spanning two columns) is its name (the role or style of the requirement, e.g. `[permission]` or `[.permission]`), optionally followed by its title (the caption of the requirement, e.g. `.Title`).
- The title of the table (spanning two columns) is its `label` attribute.
- The initial rows of the body of the table give metadata about the requirement. They include:
  - The `obligation` attribute of the requirement, if given: *Obligation* followed by the attribute value
  - The `subject` attribute of the requirement, if given: *Subject*, followed by the attribute. (In other classes of requirement, the `subject` attribute of the requirement is rendered differently; see below.) The subject attribute can be marked up as a cross-reference to another requirement given in the same document. If there are multiple values of the subject, they are semicolon delimited [added in https://github.com/metanorma/metanorma-standoc/releases/tag/v1.10.4].
  - The `inherit` attribute of the requirement, if given: *Dependency* followed by the attribute value. If there are multiple values of the subject, they are semicolon delimited.
  - The `classification` attributes of the requirement, if given: the classification tag (in capitals), followed by the classification value.
- The remaining rows of the requirement are the remaining components of the requirement, encoded as table rows instead of as a definition table (as they are by default in Metanorma).
  - These include the explicit `component` components of the requirement [added in https://github.com/metanorma/metanorma-ogc/releases/tag/v1.4.0], which capture internal components of the requirement defined in ModSpec.

    These are divided into two categories:

    - Components with a `class` attribute other than `part` are extracted in order, with the class name normalized (title case), followed by the component contents. So a

component with a `class` attribute of `conditions` will be rendered as *Conditions* followed by the component contents. In the foregoing, we have seen components defined in ModSpec: `test-purpose`, `test-method`, `test-method-type`, `conditions`, `reference`. However the block attribute syntax allows open-ended component names.

- Components with the `class` attribute `part` are extracted and presented in order: each Part is rendered as an incrementing capital letter (*A*, *B*, *C* and so on), followed by the component contents. Any cross-references to part components will automatically be labelled with the label of their parent requirement, followed by their ordinal letter.

- Components can include descriptive text (`description`), which is interleaved with other components.
- Components can include open blocks marked with role attributes. That includes the legacy Metanorma components:
  - `[.specification]`
  - `[.measurement-target]`
  - `[.verification]`
  - `[.import]`

F

ANNEX F (INFORMATIVE) CONSISTENCY IN MODELING WITH CONSTRAINTS

ANNEX F (INFORMATIVE) CONSISTENCY IN MODELING WITH CONSTRAINTS

F.1. Introduction

In recent years Sparx Systems Enterprise Architect has evolved two separate mechanisms for documenting constraints as part of a UML class diagram (conceptual model). Although superficially similar they do not result in the same outcome. Only one mechanism manipulates constraints as formal modeling elements; the other is an extension of the long-standing graphic "Note" that may be attached to any modeling element in a class diagram.

Figure F.1 illustrates a class from CityGML with three attached Notes. The class presentation includes its constraints compartment which identifies two constraints.

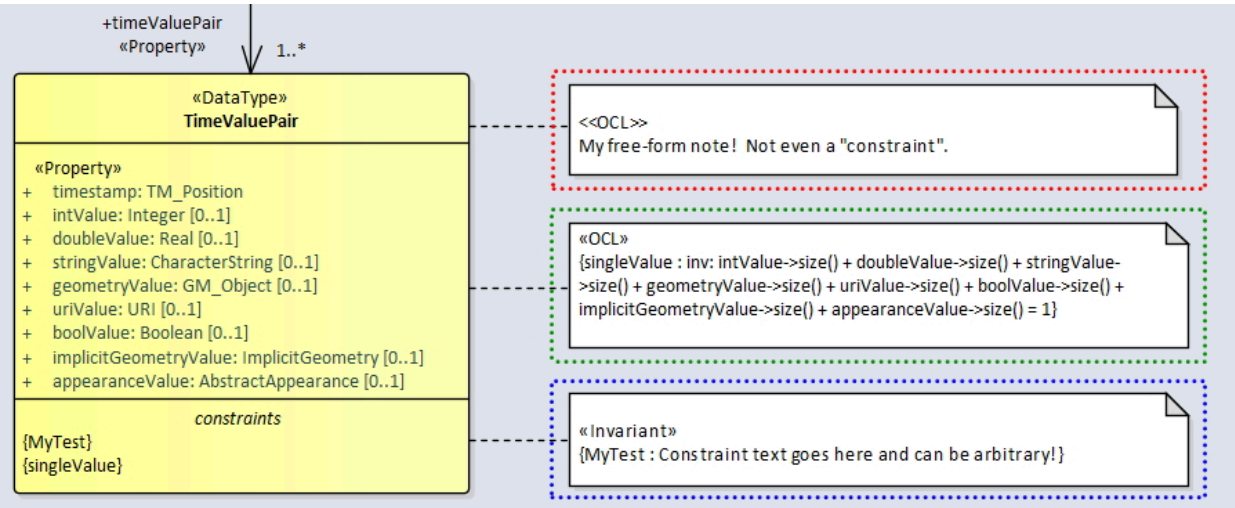


Figure F.1 — CityGML Class with Constraints and Notes

The first Note (inside the dotted red line) is a simple graphic Note whose content has been styled to appear similar to a constraint. While this is a valid presentation-oriented approach to documenting a conceptual model, the content of the Note cannot be employed in Model Driven Architecture (MDA). The Note has no associated semantics.

The second Note (inside the dotted green line) is a named formal OCL constraint “singleValue” whose syntax (but not semantics) has been validated by EA. The constraint can be employed in MDA. The visible Note is synchronized with the constraint specification and cannot be independently edited; it is presentation-only. Note that OCL-determined syntax specifies that the type of constraint be specified; the substring `inv:` specifies that an invariant constraint is being defined.

The third Note (inside the dotted blue line) is a named text-based invariant constraint “MyTest” whose content is not subject to validation by EA. The constraint may be employable in MDA if the format of the content adheres to some externally-specified language. The visible Note is synchronized with the constraint specification and cannot be independently edited; it is presentation-only.

The ISO/TC 211 Harmonized Model Maintenance Group (HMMG) conceptual schemas generally follow a fourth pattern. Figure F.2 illustrates a class from the HMMG conceptual model for ISO 19107:2019 with one attached Note. The class presentation includes its constraints compartment which identifies three constraints.

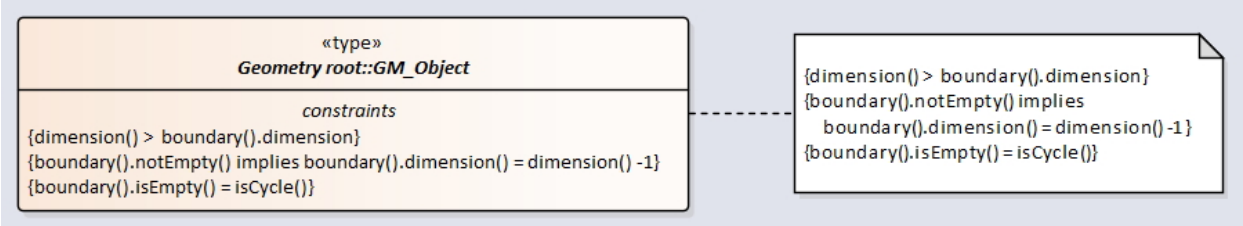


Figure F.2 — HMMG Class with Constraints and Notes

The Note is simple, including formatting that uses braces, line breaks, and padding to delineate three constraints. The individual constraints are unnamed. While this is a compact presentation-oriented approach to documenting a conceptual model, the content of the Note cannot be employed in Model Driven Architecture (MDA). The Note has no associated semantics. The visible Note has no relationship to the three class constraint specifications; it can be independently edited.

Separately there are three constraints identified in the constraints compartment of the class. They are unnamed; instead the complete body of the constraint is presented (which can be a problem for long constraints, since EA does not allow the presentation of a constraint to wrap onto multiple lines and will clip it at 255 characters). They are typed (correctly) as invariant constraints, although this information is nowhere visible in the presentation. The content of these unnamed text-based invariant constraints is not subject to validation by EA. These constraints (once it is understood that the body of the constraint has been substituted for the name) may be employable in MDA if the format of the content adheres to some externally-specified language. That is not currently the case; the choice of “constraint expression” is left to the discretion of ISO/TC 211 editing committees.

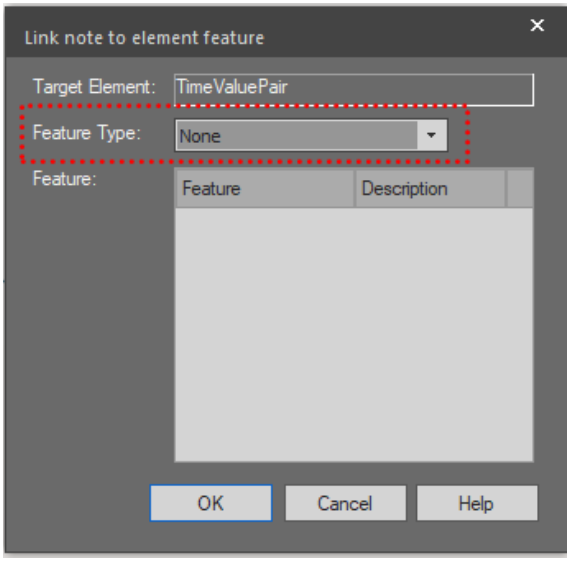
F.2. Documenting an OCL Invariant Constraint

For general information regarding the EA "Notes" capability, see: https://sparxsystems.com/enterprise_architect_user_guide/14.0/modeling_tools/noteswindow.html

In order to successfully create a named OCL invariant constraint with a synchronized Note-based presentation:

1. Create a "Note" using the Diagram Toolbox (Common section). Do not populate it.
2. Add a "Note link" using the Diagram Toolbox (Common Relationships section) from the Note to the constraint-associated class.
3. Right-click on the new link to activate a pop-up actions menu whose first entry is "Link this note to an element feature ..."; select that entry.
4. The resulting dialog lets you pick the specific element, first the "Feature Type" and then the "Feature". These two selections are illustrated in Table F.1. following diagram.

Table F.1 — Constraint Selection



5. The resulting populated Note is non-editable; a pop-up dialog gives guidance as to how to change its content (Figure F.3).

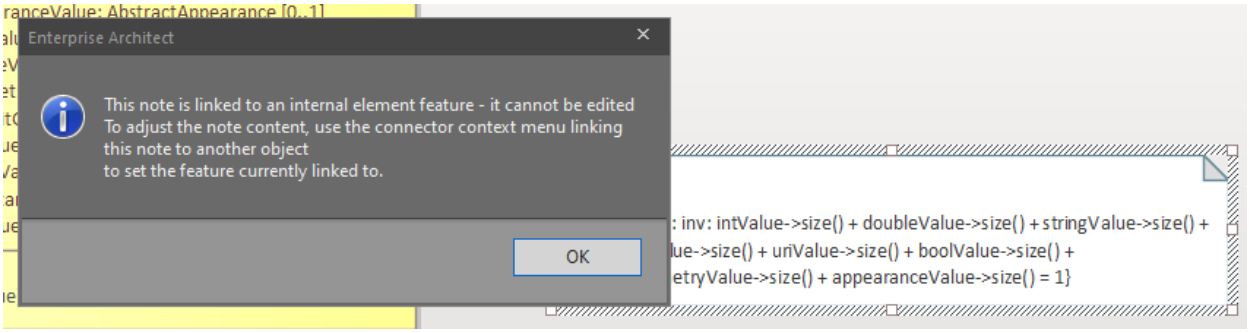


Figure F.3 — Editing a Constraint-derived Note

Editing the constraint content using the Class Editor will automatically update the Note. The constraint should always have Type = `OCL`. The EA constraint editor where this choice is determined is illustrated in Figure F.4.

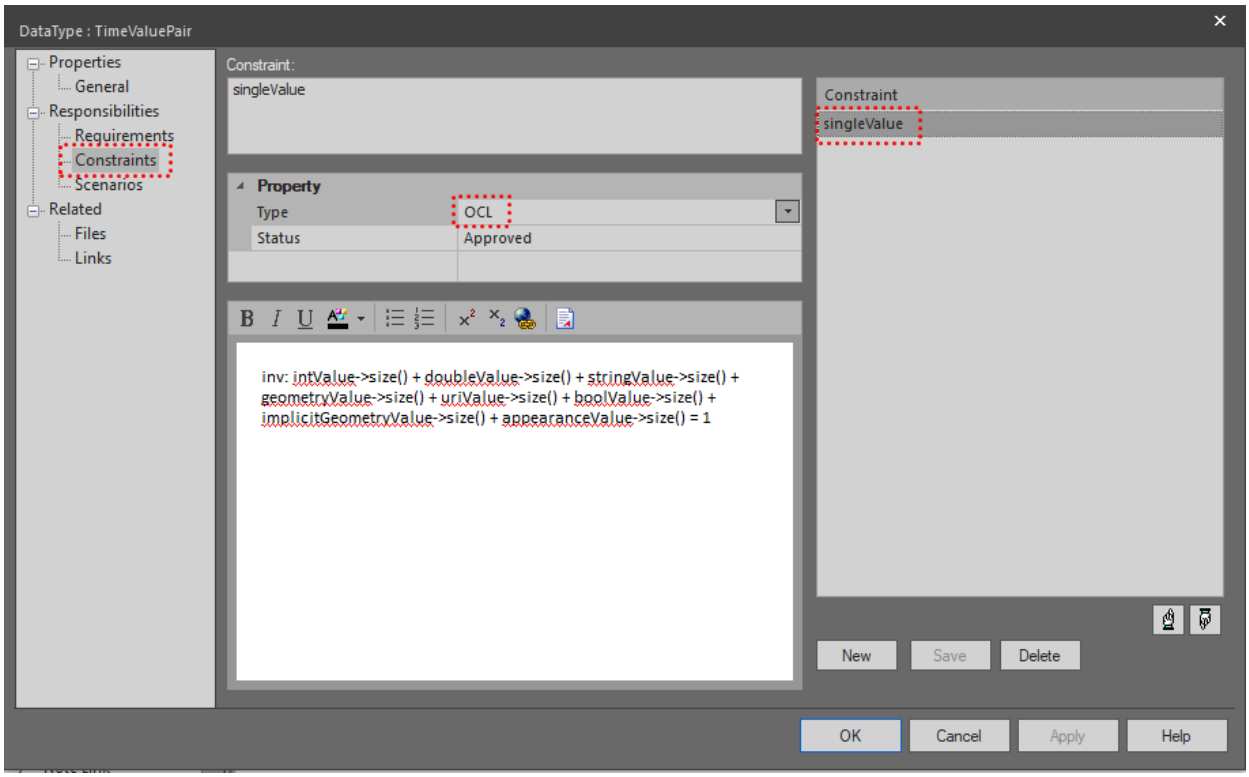


Figure F.4 — Editing an OCL Constraint

Note that while the constraint editor allows the name to be edited, the link to the Note will be broken and the Note left essentially empty (just a pair of braces will be present). The link can be easily re-established.

F.3. Documenting a Text Invariant Constraint

In order to successfully create a named text invariant constraint with a synchronized Note-based presentation, essentially the same procedure is followed as for creating an OCL invariant constraint. The principal difference is that the choice of Type should be “Invariant” rather than “OCL”. The content of the constraint can be free-text (Figure F.5).

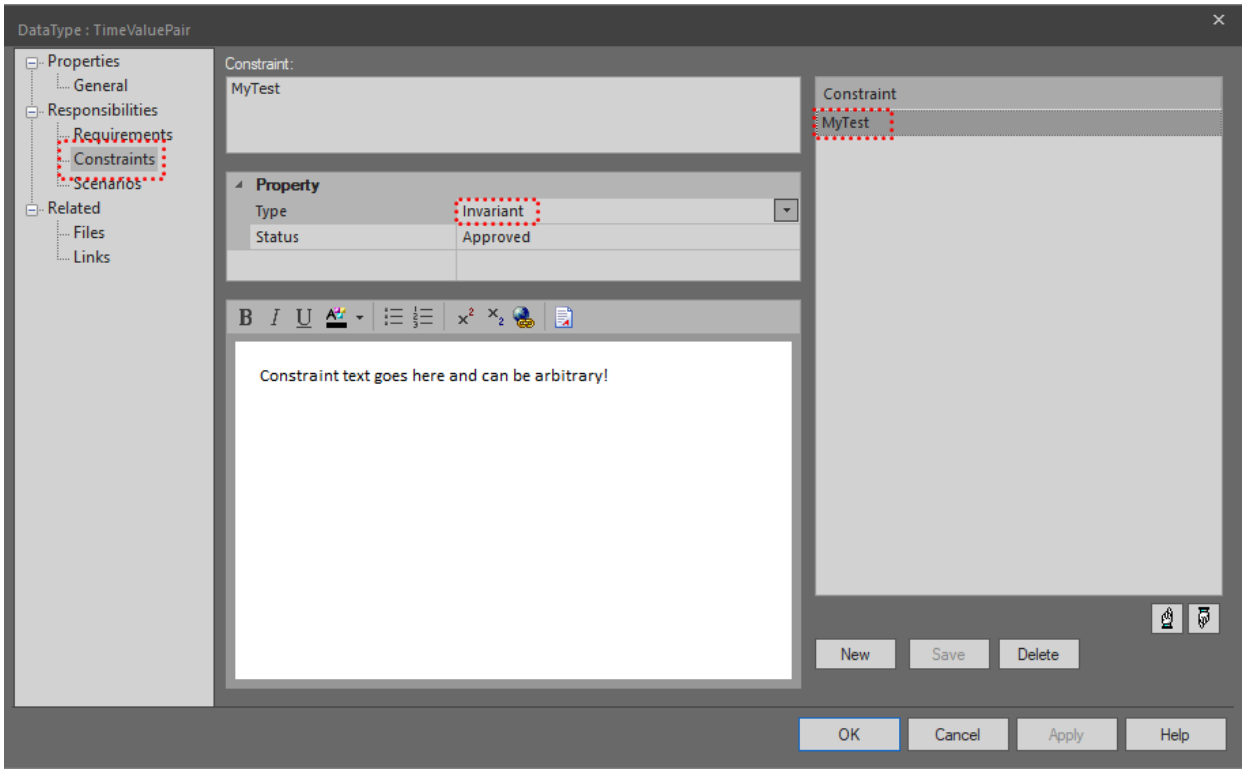


Figure F.5 — Editing a Text Constraint

If “OCL” is inadvertently selected then a pop-up warning will be issued when EA unsuccessfully attempts to parse the nominally OCL expression, as illustrated in Figure F.6.

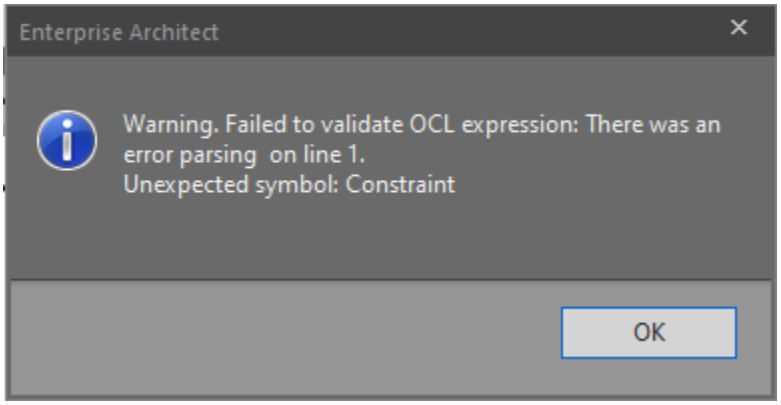


Figure F.6 — OCL Validation Failure

G

ANNEX G (INFORMATIVE) EMF REPAIR AND NORMALIZATION FROM EA-GENERATED EMFS

ANNEX G (INFORMATIVE) EMF REPAIR AND NORMALIZATION FROM EA-GENERATED EMFS

G.1. General

Microsoft EMF is a proprietary vector graphics format used on Microsoft Windows. Its usage today is more limited to the Microsoft Office suite, which does not handle any vector image format outside of EMF.

EMFs are not popular as it is not a standardized format, and does not have wide support across platforms. Microsoft in recent years have provided an EMF specification, however, the reputation is that EMFs that interoperability of EMF files do not necessarily depend on the conformance to specification.

Sparx Systems Enterprise Architect when used as a model authoring tool, can produce UML diagrams. As UML diagrams are typically drawn with lines and shapes, they are best suited to be exported as vector images. However, Sparx Systems Enterprise Architect only supports the EMF format for export, and not platform-independent vector graphics like W3C SVG.

In the standards world, vector images are preferred over raster images due to accessibility which allows the standard document to be translated without affecting the diagrams. Moreover, the ability to serve an audience across multiple platforms and formats is cherished, and thus image formats used must also be conducive to this goal.

In particular, OGC requires vector graphics to be rendered in platform-independent formats for HTML, PDF and Word outputs, and the MDS tool must handle conversions from EMF to SVG.

In this section we document issues encountered with the Sparx Systems Enterprise Architect implementation of EMF and tasks performed by Ribose Limited (for OGC Testbed-17 D144) to address them.

G.2. Issue 1: EA-generated EMFs have abnormal Y-coordinates

- EA diagrams were , treating coordinates differently (the y-axis is inverted, but text on the y-axis is not). They thus required a fork of the `libemf2svg` library () in order to be processed correctly.

While Sparx Systems Enterprise Architect generates EMF files for UML diagrams, the EA-generated EMF files are not compliant to the Microsoft EMF specification.

The Microsoft EMF specification specifies drawing coordinates in the following manner (see Figure G.1):

- the origin point (with coordinates `[0,0]`) is located at the upper-left corner of the window;
- x-coordinates increase to the right
- y-coordinates increase from bottom to top.

The SVG coordinate system, by default, uses the same origin point (`[0,0]`) as does EMF at the upper left corner of the window, with identical behavior of x-coordinate and y-coordinates increasing from top to bottom.

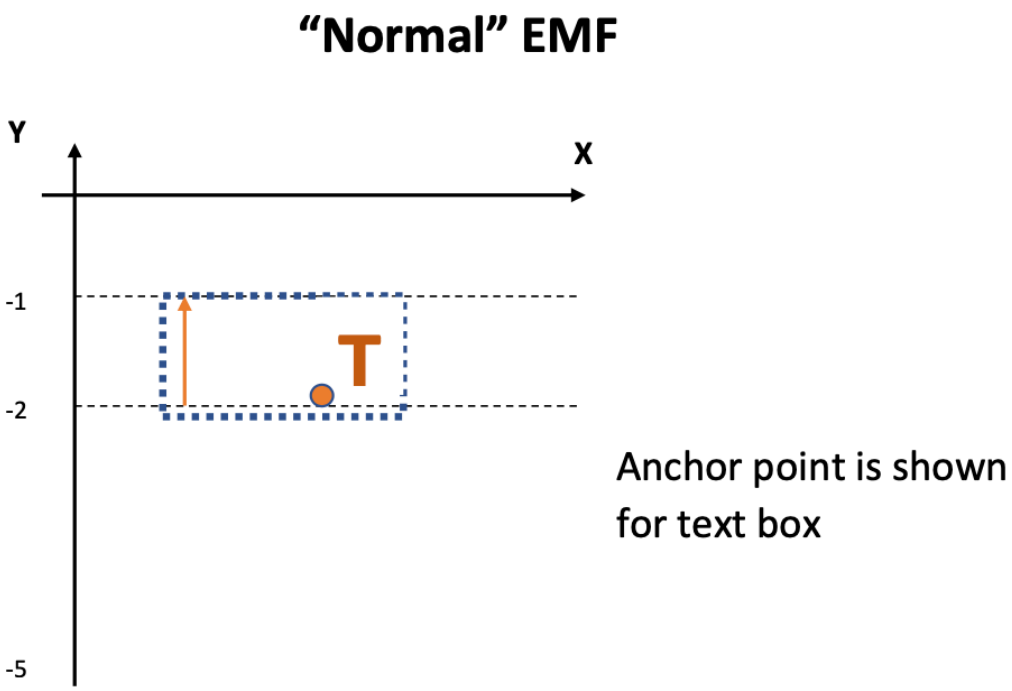


Figure G.1 — Illustration of coordinates in normal EMF

The EMF images generated by Sparx Systems Enterprise Architect, however, sport malformed coordinates (see Figure G.2):

- the origin point (`[0,0]`) is still located at the upper-left corner
- x-coordinates still increase to the right
- but y-coordinates are inverted (multiplied by `-1`).

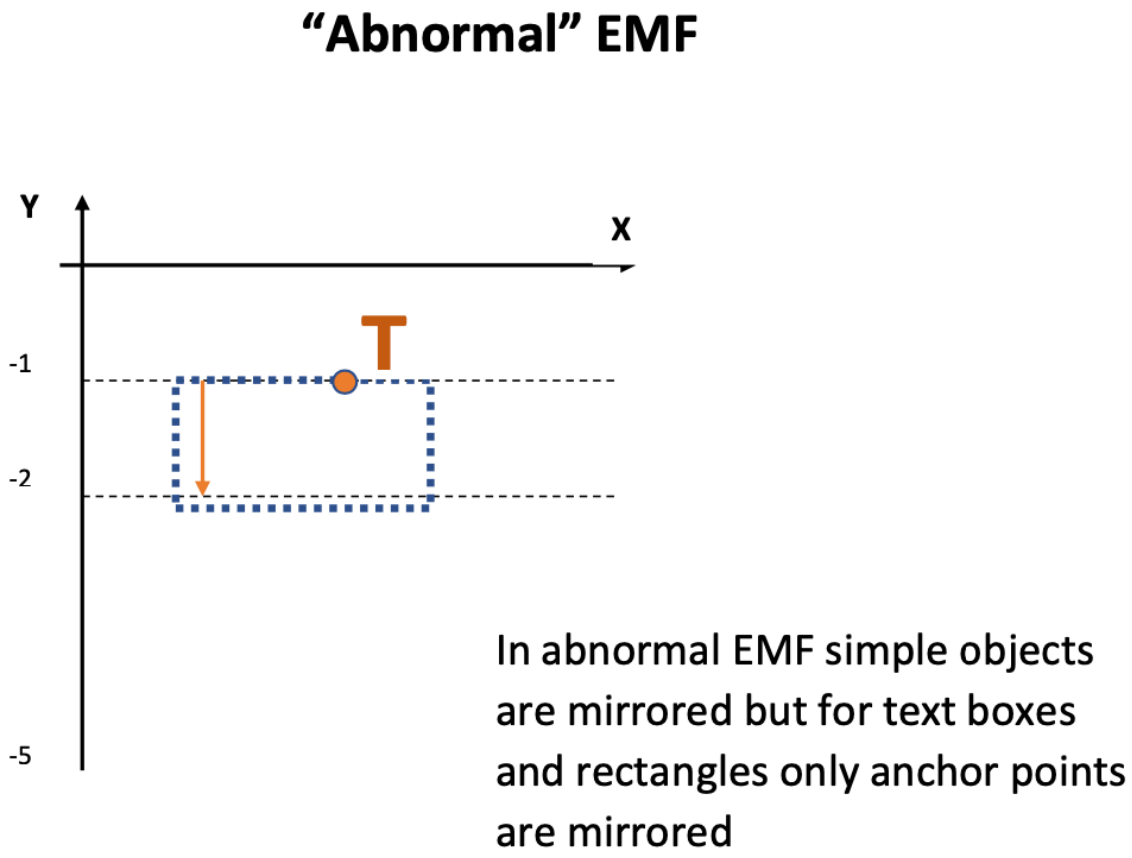


Figure G.2 — Illustration of coordinates in EA-generated EMF

Furthermore, with complex objects, this inversion is only a partial mirroring of an object.

For example, text boxes have only their y-coordinate of the anchor point mirrored, but the text direction is not changed.

Interestingly, Microsoft Word's EMF rendering engine fixes up the broken y-coordinates, as seen in Figure G.3, but other EMF tools such as Adobe Illustrator and Inkscape remain faithful to the input, such as Figure G.4.

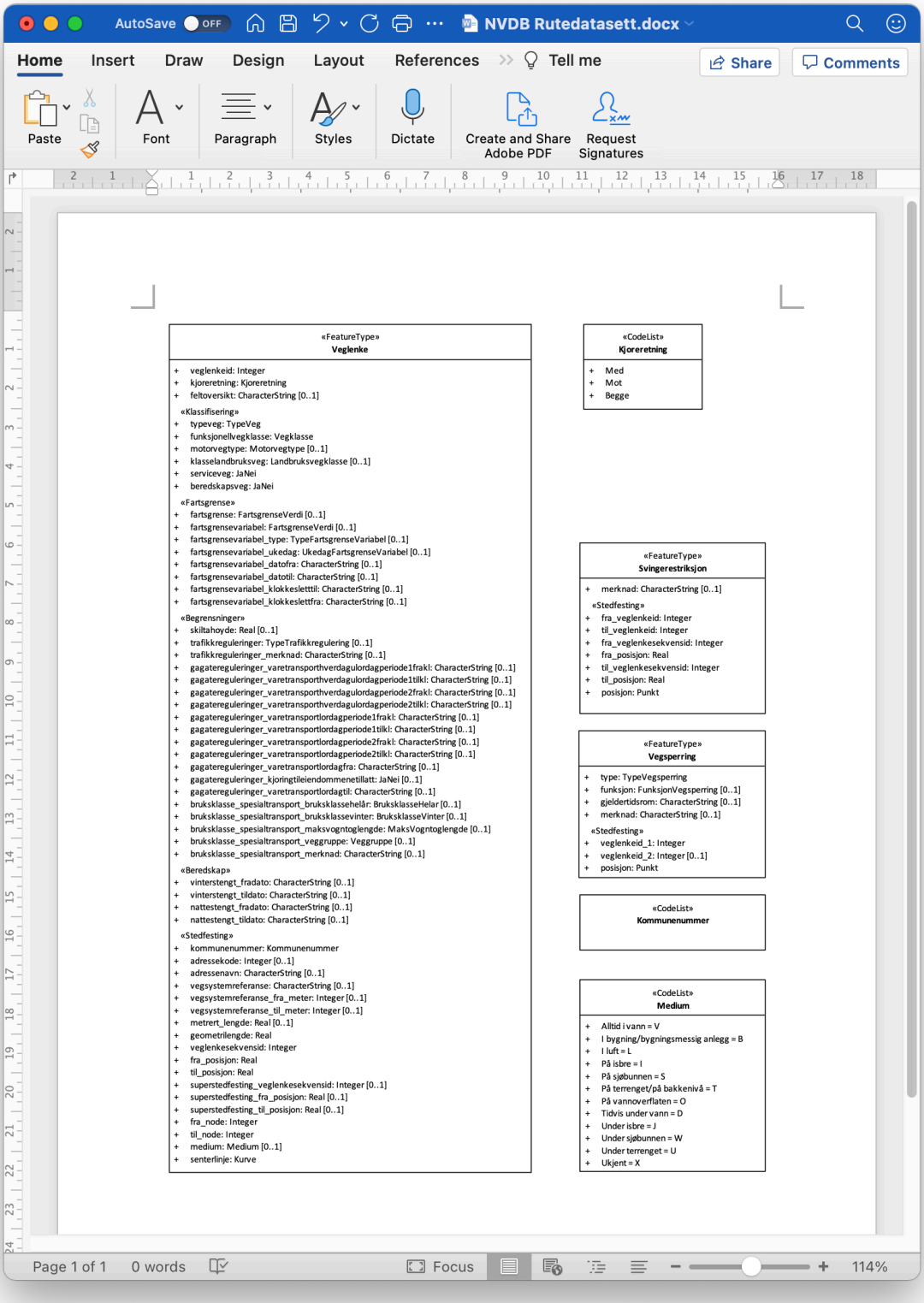


Figure G.3 — EA-generated EMFs displays properly in Microsoft Word

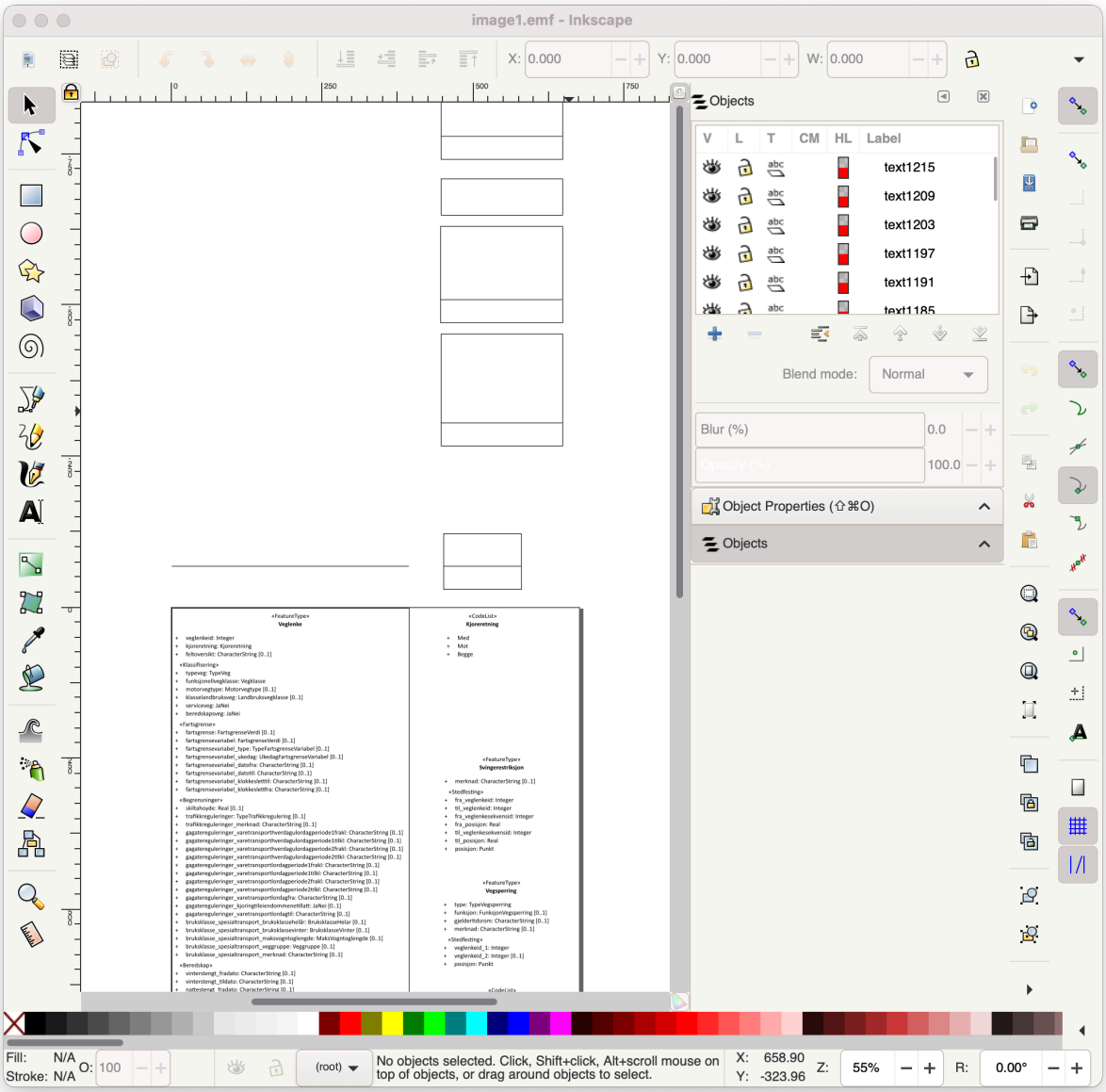


Figure G.4 — EA-generated EMFs does not render properly in non-Microsoft EMF-compliant tools (Inkscape shown)

This specific layout problem cannot be fixed by any single SVG/CSS transformation operation (see Figure G.5).

In order to address this issue, Ribose has forked and updated the `libemf2svg` library (https://github.com/metanorma/libemf2svg) in order to fix EA-generated EMFs, by detecting such condition and repairing the EMF coordinates. The resulting repair transformation is shown in Figure G.6.

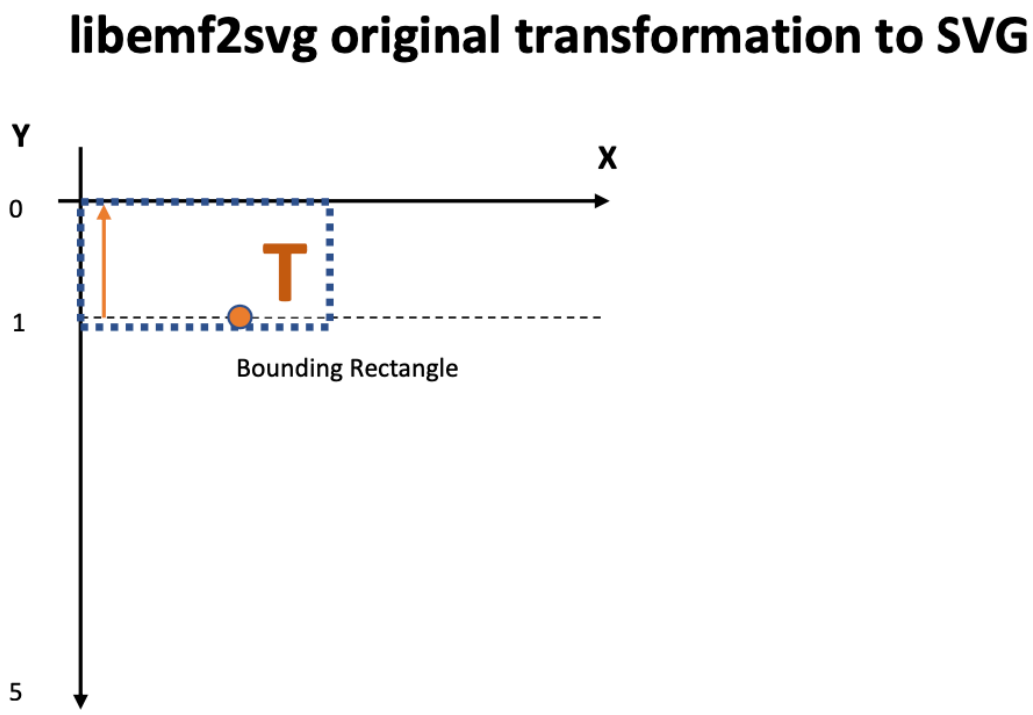


Figure G.5 — Standard transformation from EMF to SVG

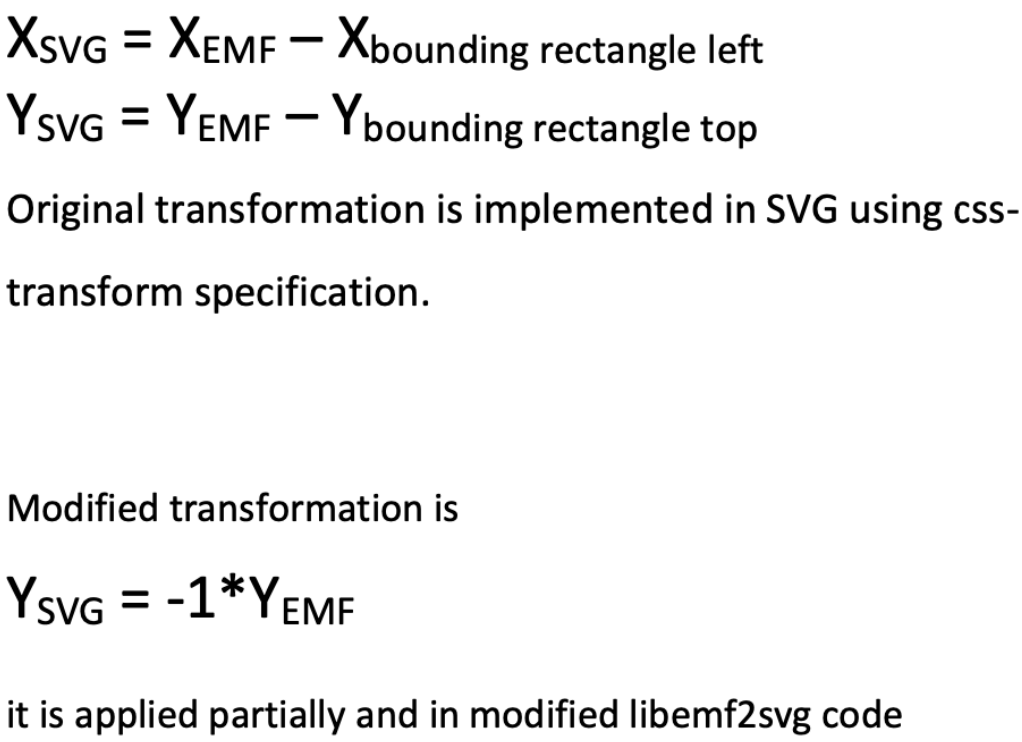


Figure G.6 — Repair transformation from EMF to SVG to normalize flipped coordinates

The `libemf2svg` library is integrated into Metanorma, and therefore this issue is fully addressed in an automated fashion.

G.3. Issue 2: EMFs lacking viewport when generated using EA on Wine

Sparx Systems Enterprise Architect is a 32-bit Windows application that is tightly bound to the Windows platform. As a result, running Sparx Systems Enterprise Architect on other platforms requires installation of the free and open source Wine product, or the commercial CrossOver product.

The EMF functionality on Sparx Systems Enterprise Architect relies on GDI+, which is a "class-based API for C/C++ programmers" that can generate EMF. However, the GDI+ library is known to be problematic when running on Wine.

The particular problem is that EMF images will be shown to be "blank", and this applies to both Windows and non-Windows platforms. The EMF files generated on Sparx Systems Enterprise Architect via Wine are just seemingly blank even though the contents seem to be present.

This issue has been acknowledged by Sparx Systems at least since 2016, and it is documented at its support forum:

- https://sparxsystems.com/forums/smf/index.php/topic,37405.0.html

The workaround for this problem, as reported by Sparx Systems support staff, was to fall back from GDI+ to GDI. As of Sparx Systems Enterprise Architect version 15.1, it is supposedly possible to turn off GDI+.

- https://www.sparxsystems.com.au/enterprise_architect_user_guide/15.1/user_interface/diagram_appearance.html , as shown in Figure G.7

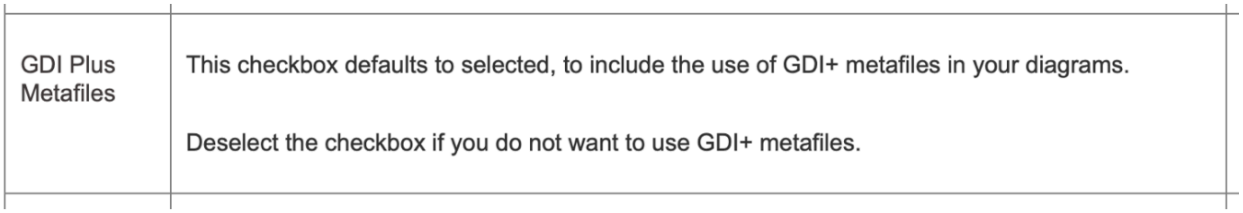


Figure G.7 — Enterprise Architect User Guide for v15.1 allowing for disabling of GDI+

However, the testbed participants found that the option to disable GDI+ is not present in Sparx Systems Enterprise Architect version 15.1. This is an inconsistency between the application itself and the user guide claiming unavailable functionality. The testbed participants suspect that the option was present in version 14 but removed in 15, and the user guide was not updated to reflect this change.

Since falling back to GDI is no longer possible, the testbed participants investigated the issue in depth.

The reasons why the EA-generated EMF files are blanked out:

1. Text objects in these EMFs have been set with font size `0`. This is clearly shown in an SVG converted from the EMF.

   Example :

```
<text font-family="Carlito" fill="#000000" style ="white-space:
pre;"
font-weight="700" text-anchor="start" x="0.0000" y="0.
0000"
font-size="0.0000" ><![CDATA[SF SamplingSolid]]></text>
```

2. All font attributes are set to value `0` because the viewport `viewPortExX` property is always `0`.

3. The `viewport` property is never initialized since there are no `EMR_SETVIEWPORTEXTEX` records in EMF files.

4. The root cause of the problem is that a wrong default value was given to `viewPortExx`.

Ribose implemented the fix to the EMF Viewport during the EMF-to-SVG conversion flow in the forked version of `libemf2svg`.

The `libemf2svg` library is integrated into Metanorma, and therefore this issue is fully addressed in an automated fashion.

G.4. Issue 3: EA-generated EMFs lack graphical details that exist in EA-generated PNGs

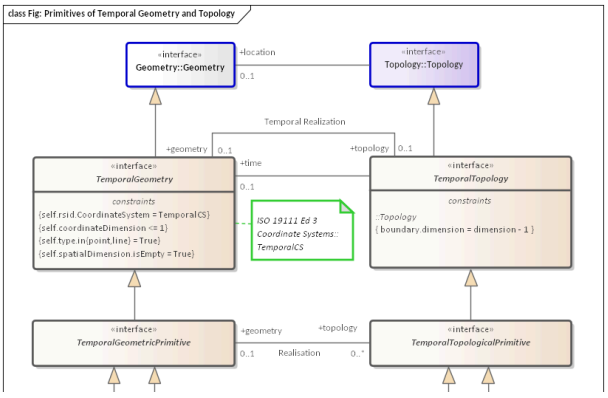
This is an issue that is purely presentational, and that the Testbed participants decided not to pursue a fix for. It does however highlight the inconsistency of graphics generated by Sparx Systems Enterprise Architect.

In the PNG output of UML diagrams, a gradient color shading is generally applied to all UML graphic blocks for an increased perception of depth.

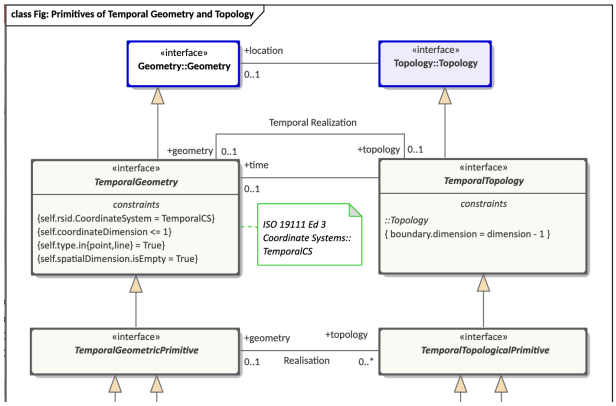
In the corresponding EMF output, those gradients are all missing. Upon investigation, the shapes that contain those gradients were present but the color gradient is not present.

Since this is a purely presentational issue, and that the color gradient actually somewhat impairs readability, no fix is implemented for this issue.

Table G.1 — Comparing EA-generated PNG and EMF renderings



EA-generated PNG file of a DGGS diagram



EA-generated EMF file of a DGGS diagram

H

ANNEX H (INFORMATIVE) KNOWLEDGE GRAPH V. PROPERTY GRAPHS

ANNEX H
(INFORMATIVE)
KNOWLEDGE GRAPH V. PROPERTY GRAPHS

> I want to have a relationship between classes, say *Association*, but I want to be able to add extra facts about that relationship. Is this a Property Graph thing?

H.1. Property Graphs

Consist of:

- Nodes
- Edges
- Properties
  - key/value pairs for information, or “tags”
  - can be associated with Nodes or Edges

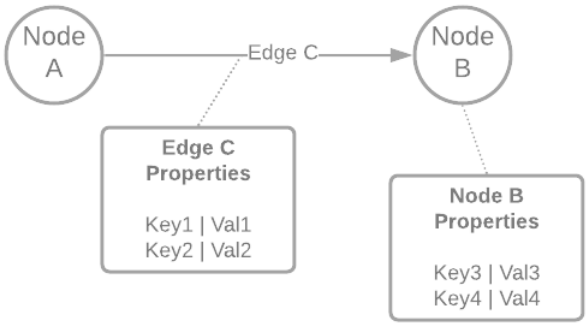


Figure H.1 — Basic components of a Property Graph

So yes, you can easily represent a relationship `-C→` as an Edge between things `(A)` & `(B)` (Nodes) in a Property Graph (PG) and associate anything you like with Edge `-C→` as PG Properties. However:

- How is the Properties information in the PG formalized?
  - PGs do not offer a consistent standard for doing this
  - Even when the main PG content — Nodes & Edges — are formally defined, the Properties (the keys & values) may not be
- How are instances of data, claiming conformance to the PG, implemented and tested?
  - PGs do not offer a standard way of interpreting PG information as rules that can be used for data validation

H.2. Knowledge Graphs

Knowledge Graphs (KGs) can represent the information in Figure H.1 in several ways with strong definitions and executable validation rules. There are two main ways to do this in RDF, called the *Qualified Relationship* and the *Reification* pattern. There is a 3rd way too which is a variant of Reification called *RDF**. These patterns are described below.

H.2.1. Qualified Relationship Pattern

RDF-based KGs are made of only Nodes and Edges and ensure that everything in them is a universally-defined Node or Edge. To emulate the `(A) -C→ (B)` structure in Figure H.1 with more information about `C`, we could equate the relationship, a PG Edge, `-C→` to a KG Edge + Node + Edge like this:

`-C-> == -X-> (Y) -Z->`

Figure H.2

So `(A) -C→ (B)` becomes `(A) -[-X→ (Y) -Z→]→ (B)`. Then we can associate anything we like with the intermediate Node `(Y)`. This is shown in Figure H.3.

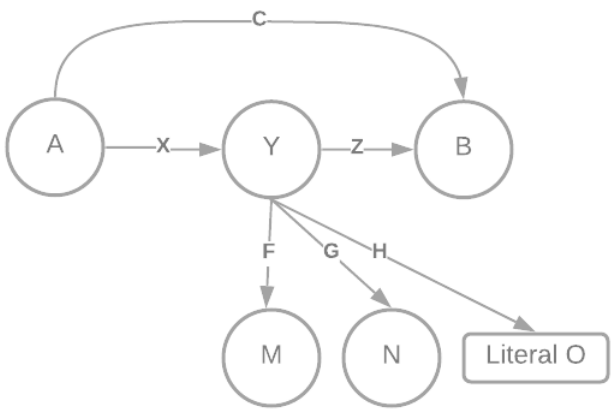


Figure H.3 — Qualified Relationship Pattern equivalent to Figure H.1 with an intermediate Node, `(Y)` being placed between `(A)` & `(B)`

Other things are associated with `(Y)`, here relationships, `-F→`, `-G→` & `-H→` to yet other nodes, `(M)` & `(N)` and to a literal (text, number etc.) to 'qualify' the original relationship `-C→`.

This Qualified Relationship pattern is the normal/most common way that a KG uses to add more information to/about a relationship and, in RDF/OWL, any set of relationships, and the intermediate nodes, can be equated to another relationship formally using a *property chain axiom*. Using such, the relationship equivalence of `-C→ == -X→ (Y) -Z→` would formally be defined like this (Turtle syntax RDF):

```
example:C owl:propertyChainAxiom ( example:X example:Z ) .
```

To ensure that every such chain includes a node of type *Y* we then add:

```
example:X rdfs:range example:Y .
```

So now whenever we see `example:C`, it is equal to `example:X` then `example:Z` with the `example:X` property requiring the intermediate node to be of type *Y*, and this would be required of either named or Blank (un-identified) intermediate nodes.

Finally, we can impose any RDF/OWL restrictions on nodes of type *Y* that we like, for instance insisting that they indicate properties to other nodes (classes or individuals) or simple data properties:

```
example:Y
    a owl:Class ;
    rdfs:subClassOf [
        a owl:Restriction ;
            owl:onProperty example:F ;
            owl:allValuesFrom example:M ;
            owl:cardinality 1 ;
        ] ;
.
```

The OWL fragment above states that "every instance of Class `Y` must have a property `F` that indicates one and only one instance of Class `M`."

H.2.2. Reification Pattern

Reification allows us to make meta-statements about other statements in an RDF graph. With this pattern, we need not change the basic association `(A) –C→ (B)` but talk about it elsewhere.

Any triple in an RDF store is composed of a *subject*, *predicate* & *object* and the fundamental structural ontology for RDF also defines a `Statement` class of object which has, as its expected properties, `rdf:subject`, `rdf:predicate` & `rdf:object` which can be used to indicate the elements of any other triple. So, we can create a `Statement`, `S`, which indicates `(A) –C→ (B)` and then says anything else we want to about it, as per Figure H.4.

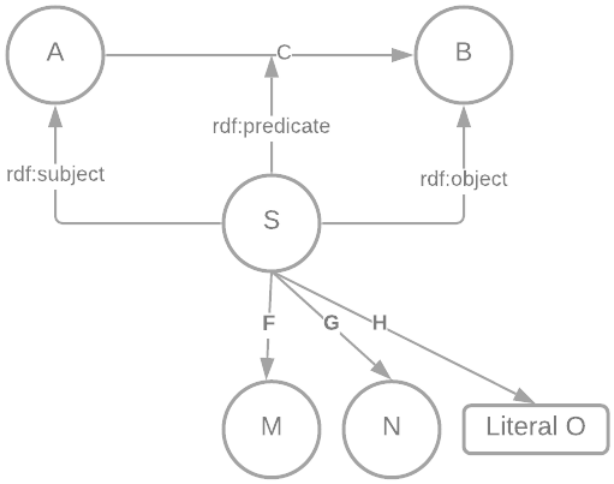


Figure H.4 — Reification of the elements in Figure H.1

Statement `S` indicates `A`, `B` & `C` and other things, such as `M`, `N`, and literal `O`, just like in the qualified pattern in Figure H.3.

Reification is often used when primary facts need secondary facts/meta facts/provenance recorded about them semi-separately to the original data. However, if Statement `S` is stored in the same store as `(A) –C→ (B)` then we can query for the elements of the statement easily.

The full RDF code for Figure H.4 is:

```
example:A example:C example:B .

example:S
    a rdf:Statement ;
    rdf:subject example:A ;
    rdf:predicate example:C ;
    rdf:object example:B ;
    example:F example:M ;
    example:G example:N ;
    example:H "Some Literal O" ;
.
```

...and any other statements for restrictions.

Reification is often used for things like uncertainty or quality *of the statement referred to*, e.g.:

```
example:S
    a rdf:Statement ;
    rdf:subject example:A ;
    rdf:predicate example:C ;
    rdf:object example:B ;
    example:certainty 0.75 ;
.
```

where 0.75 is the "certainty" of the statement `example:A example:C example:B .` being true.

H.2.3. RDF* Pattern

RDF* is an extension to RDF that is supported by a number of vendors' productions, including major triplestores like Jena and GraphDB. It is likely to be formalized in a W3C standard within the next 12 months.

RDF* really just proposes, per RDF syntax (JSON-LD, Turtle, RDF/XML, N-triples) a more compact way of showing reification. So, for the reification in Figure H.4 where `(A) –C→ (B)` is represented by the RDF triple `example:A example:C example:B .`, you would write this, in Turtle*:

```
 <<example:A example:C example:B>>
    example:F example:M ;
    example:G example:N ;
    example:H "Some Literal O" ;
.
```

So RDF* does no more than Reification but presents it better!

Triplestores that support RDF* and SPARQL* allow compact notation using << & >>, as above in data and queries and convert that, internally, to reified information.

H.3. Conclusion

Knowledge Graphs contain functional equivalents to Property Graph Properties for Edges which are in widespread use. RDF* was created recently specifically to help people familiar with PGs start to use KGs.

The big KG benefit over PGs is the retention of semantics for the Properties, not just Nodes and Edges, which mean:

- no magic properties in KGs
    - what does *KeyX* in a PG Property table actually mean?

- standard query language
    - SPARQL can query *Qualified Patter* or *Reified Pattern* and extended PSARQL, SPARQL* can query *RDF* Pattern* data
- standard rules & validation
    - OWL rules, SPARQL ASK statements and Shapes languages such as SHACL all work happily with any of the 3 patterns

I

ANNEX I (INFORMATIVE) REVISION HISTORY

I ANNEX I (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2022-02-10	0.1	Ronald Tse, Nick Nicholas	all	initial version

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Policy SWG: OGC 08-131r3, *The Specification Model — Standard for Modular specifications*. Open Geospatial Consortium (2009). https://portal.ogc.org/files/?artifact_id=34762&version=2

2. Thomas H. Kolbe, Tatjana Kutzner, Carl Stephen Smyth, Claus Nagel, Carsten Roensdorf, Charles Heazel: OGC 20-101, *OGC CityGML Part 1*. Open Geospatial Consortium (2021). https://docs.ogc.org/is/20-010/20-010.html

3. Johannes Echterhoff: OGC 20-012, *UML-to-GML Application Schema Pilot (UGAS-2020) Engineering Report*. Open Geospatial Consortium (2021). https://docs.ogc.org/per/20-012.html

4. Johannes Echterhoff: OGC 18-032r2, *OGC Testbed-14: Application Schema-based Ontology Development Engineering Report*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/per/18-032r2.html

5. Johannes Echterhoff: OGC 18-091r2, *OGC Testbed-14: Application Schemas and JSON Technologies Engineering Report*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/per/18-091r2.html

6. Clemens Portele: OGC 04-100, *OWS-2 Application Schema Development*. Open Geospatial Consortium (2005). https://portal.ogc.org/files/?artifact_id=8071

7. Gerhard Gröger, Thomas H. Kolbe, Claus Nagel, Karl-Heinz Häfele: OGC 12-019, *OGC City Geography Markup Language (CityGML) Encoding Standard*. Open Geospatial Consortium (2012). https://portal.ogc.org/files/?artifact_id=47842

8. Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r3, *OGC API — Features — Part 1: Core*. Open Geospatial Consortium (2019). http://docs.opengeospatial.org/is/17-069r3/17-069r3.html

9. Robert Gibb: OGC 20-040r3, *OGC Topic 21 — Discrete Global Grid Systems — Part 1 Core Reference system and Operations and Equal Area Earth Reference System*. Open Geospatial Consortium (2021). https://docs.ogc.org/as/20-040r3/20-040r3.html

10. Hugo Ledoux: OGC 20-072r2, *OGC CityJSON Community Standard 1.0*. Open Geospatial Consortium (2021). https://docs.ogc.org/cs/20-072r2/20-072r2.html

11. ISO: ISO 690, *Information and documentation — Guidelines for bibliographic references and citations to information resources*. International Organization for Standardization, Geneva https://www.iso.org/standard/72642.html

12. ISO: ISO 704, *Terminology work — Principles and methods*. International Organization for Standardization, Geneva https://www.iso.org/standard/38109.html

13. ISO: ISO 8601-1, *Date and time — Representations for information interchange — Part 1: Basic rules*. International Organization for Standardization, Geneva https://www.iso.org/standard/70907.html

14. ISO: ISO 10303-11, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual*. International Organization for Standardization, Geneva https://www.iso.org/standard/38047.html

15. ISO: ISO 19109:2015, *Geographic information — Rules for application schema*. International Organization for Standardization, Geneva (2015). https://www.iso.org/standard/59193.html

16. ISO: ISO 19101-1:2014, *Geographic information — Reference model — Part 1: Fundamentals*. International Organization for Standardization, Geneva (2014). https://www.iso.org/standard/59164.html

17. ISO: ISO 19107:2019, *Geographic information — Spatial schema*. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/66175.html

18. ISO: ISO 19107:2003, *Geographic information — Spatial schema*. International Organization for Standardization, Geneva (2003). https://www.iso.org/standard/26012.html

19. ISO: ISO 19108:2002, *Geographic information — Temporal schema*. International Organization for Standardization, Geneva (2002). https://www.iso.org/standard/26013.html

20. ISO: ISO 19111:2019, *Geographic information — Referencing by coordinates*. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/74039.html

21. ISO: ISO 19112:2019, *Geographic information — Spatial referencing by geographic identifiers*. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/70742.html

22. ISO: ISO 19111:2007, *Geographic information — Spatial referencing by coordinates*. International Organization for Standardization, Geneva (2007). https://www.iso.org/standard/41126.html

23. ISO: ISO 19115-1:2014, *Geographic information — Metadata — Part 1: Fundamentals*. International Organization for Standardization, Geneva (2014). https://www.iso.org/standard/53798.html

24. ISO: ISO 19123:2005, *Geographic information — Schema for coverage geometry and functions*. International Organization for Standardization, Geneva (2005). https://www.iso.org/standard/40121.html

25. ISO: ISO 19136:2007, *Geographic information — Geography Markup Language (GML)*. International Organization for Standardization, Geneva (2007). https://www.iso.org/standard/32554.html

26. ISO: ISO 19136-1:2020, *Geographic information — Geography Markup Language (GML) — Part 1: Fundamentals*. International Organization for Standardization, Geneva (2020). https://www.iso.org/standard/75676.html

27. ISO: ISO 19136-2:2015, *Geographic information — Geography Markup Language (GML) — Part 2: Extended schemas and encoding rules*. International Organization for Standardization, Geneva (2015). https://www.iso.org/standard/61585.html

28. ISO: ISO 19150-2, *Geographic information — Ontology — Part 2: Rules for developing ontologies in the Web Ontology Language (OWL)*. International Organization for Standardization, Geneva https://www.iso.org/standard/57466.html

29. ISO: ISO 19156:2011, *Geographic information — Observations and measurements*. International Organization for Standardization, Geneva (2011). https://www.iso.org/standard/32574.html

30. ISO: ISO 19170-1:2021, *Geographic information — Discrete Global Grid Systems Specifications — Part 1: Core Reference System and Operations, and Equal Area Earth Reference System*. International Organization for Standardization, Geneva (2021). https://www.iso.org/standard/32588.html

31. ISO: ISO 10241-1, *Terminological entries in standards — Part 1: General requirements and examples of presentation*. International Organization for Standardization, Geneva https://www.iso.org/standard/40362.html

32. ISO: ISO 19103:2015, *Geographic information — Conceptual schema language*. International Organization for Standardization, Geneva (2015). https://www.iso.org/standard/56734.html

33. ISO: ISO/TS 19103:2005, *Geographic information — Conceptual schema language*. International Organization for Standardization, Geneva (2005). https://www.iso.org/standard/37800.html

34. ISO: ISO 19105, *Geographic information — Conformance and testing*. International Organization for Standardization, Geneva https://www.iso.org/standard/26010.html

35. ISO: ISO 19110, *Geographic information — Methodology for feature cataloguing*. International Organization for Standardization, Geneva https://www.iso.org/standard/57303.html

36. ISO: ISO 19115:2003, *Geographic information — Metadata*. International Organization for Standardization, Geneva (2003). https://www.iso.org/standard/26020.html

37. ISO: ISO 19118:2011, *Geographic information — Encoding*. International Organization for Standardization, Geneva (2011). https://www.iso.org/standard/44212.html

38. ISO: ISO 19157:2013, *Geographic information — Data quality*. International Organization for Standardization, Geneva (2013). https://www.iso.org/standard/32575.html

39. ISO: ISO/TS 19139, *Geographic information — Metadata — XML schema implementation*. International Organization for Standardization, Geneva https://www.iso.org/standard/32557.html

40. ISO/IEC: ISO/IEC 19501, *Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*. International Organization for Standardization, International Electrotechnical Commission, Geneva https://www.iso.org/standard/32620.html

41. ISO/IEC: ISO/IEC 19507:2012, *Information technology — Object Management Group Object Constraint Language (OCL)*. International Organization for Standardization, International Electrotechnical Commission, Geneva (2012). https://www.iso.org/standard/57306.html

42. ISO/IEC: ISO/IEC 19757-3, *Information technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based validation using Schematron*. International Organization for Standardization, International Electrotechnical Commission, Geneva https://www.iso.org/standard/74515.html

43. ISO/IEC: ISO/IEC 26300, *Information technology — Open Document Format for Office Applications (OpenDocument) v1.0*. International Organization for Standardization, International Electrotechnical Commission, Geneva https://www.iso.org/standard/43485.html

44. ISO/IEC: ISO/IEC 29500 (all parts), *Information technology – Document description and processing languages*. International Organization for Standardization, International Electrotechnical Commission, Geneva (2016). https://www.iso.org/standard/71691.html

45. ISO: ISO 19160-1:2015, *Addressing — Part 1: Conceptual model*. International Organization for Standardization, Geneva (2015). https://www.iso.org/standard/61710.html

46. ISO: ISO/AWI 36100, *Standardization documents — Document metamodel*. International Organization for Standardization, Geneva https://www.iso.org/standard/77056.html

47. ISO: ISO 36200, *Standardization documents — Metadata. International Organization for Standardization, Geneva*. ISO

48. ISO: ISO 36300, *Standardization documents — Representation in XML. International Organization for Standardization, Geneva*. ISO

49. T. Bray: RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*. Internet Engineering Task Force (2014). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.7159.xml

50. H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub: RFC 7946, *The GeoJSON Format*. Internet Engineering Task Force (2016). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.7946.xml

51. T. Bray: RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*. Internet Engineering Task Force (2017). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.RFC.8259.xml

52. Austin Wright, Henry Andrews, Ben Hutton, Greg Dennis: Internet-Draft draft-bhutton-json-schema-00, *JSON Schema: A Media Type for Describing JSON Documents*. (2020). https://raw.githubusercontent.com/relaton/relaton-data-ietf/master/data/reference.I-D.bhutton-json-schema.xml

53. OMG UML 2.5, *Unified Modeling Language*. (2015). https://www.omg.org/spec/UML/2.5/About-UML/

54. OMG XMI 2.5, *XML Metadata Interchange*. Object Management Group (2015). https://www.omg.org/spec/XMI/2.5.1/About-XMI/

55. W3C TR vocab-adms, *Asset Description Metadata Schema (ADMS)*, Working Group Note. World Wide Web Consortium (2013). https://www.w3.org/TR/vocab-adms/

56. W3C XML Names, *Namespaces in XML 1.0* (Third Edition). World Wide Web Consortium (2009). https://www.w3.org/TR/xml-names/

57. W3C TR dx-connegp, *Content Negotiation by Profile*, W3C Recommendation. World Wide Web Consortium (2020). https://w3c.github.io/dx-connegp/connegp/

58. W3C TR dx-prof, *The Profiles Vocabulary*, W3C Working Group Note. World Wide Web Consortium (2019). https://www.w3.org/TR/dx-prof/

59. W3C TR skos-reference, *SKOS Simple Knowledge Organization System Reference*. World Wide Web Consortium (2009). https://www.w3.org/TR/skos-reference/

60. W3C TR shacl, *Shapes Constraint Language (SHACL)*. World Wide Web Consortium (2017). https://www.w3.org/TR/shacl/

61. W3C TR sparql11-query, *SPARQL 1.1 Query Language*. World Wide Web Consortium (2013). https://www.w3.org/TR/sparql11-query/

62. *OGC GeoSPARQL 1.1*. Open Geospatial Consortium (2021). https://opengeospatial.github.io/ogc-geosparql/

63. W3C TR turtle, *RDF 1.1 Turtle — Terse RDF Triple Language*, W3C Recommendation. World Wide Web Consortium (2014). https://www.w3.org/TR/turtle/

64. W3C TR xmlschema-1, *XML Schema Part 1: Structures* (Second Edition). World Wide Web Consortium (2004). https://www.w3.org/TR/xmlschema-1/

65. W3C TR xmlschema-2, *XML Schema Part 2: Datatypes* (Second Edition). World Wide Web Consortium (2004). https://www.w3.org/TR/xmlschema-2/

66. W3C TR annotation-model, *Web Annotation Data Model*. World Wide Web Consortium (2017). https://www.w3.org/TR/annotation-model/

67. W3C RDF, *Resource Description Framework*. World Wide Web Consortium (2014). https://www.w3.org/RDF/

68. W3C OWL. *Web Ontology Language (OWL)*. World Wide Web Consortium (2012). https://www.w3.org/OWL/

69. *OpenAPI Specification 3.0*. OpenAPI Initiative (OAI) (2020). http://spec.openapis.org/oas/v3.0.3

70. *OGC CityGML 3.0 Conceptual Model*. Open Geospatial Consortium (2021). https://github.com/opengeospatial/CityGML-3.0CM/

71. OGC Testbed-17 D023 UML Modeling Best Practices, *UML-Modeling-Best-Practices*. Open Geospatial Consortium (2022). https://gitlab.ogc.org/ogc/T17-D023-OGC-UML-Modeling-Best-Practices

72. Ribose Inc. *Metanorma*. https://www.metanorma.org

73. Ribose Inc. *Metanorma for OGC*. https://www.metanorma.org/author/ogc/

74. *CityJSON*, https://www.cityjson.org

75. Ribose Inc. *Ribose LutaML*. https://www.lutaml.org

76. *PlantUML*. https://www.plantuml.org

77. interactive instruments GmbH. *ShapeChange*. Available at: https://shapechange.net.

78. *JSON Schema*. https://json-schema.org/

79. Marco Console, Domenico Lembo, Valerio Santarelli, Domenico Fabio Savio. (2014). *Graphol: Ontology Representation Through Diagrams*. doi:10.13140/2.1.3838.3363.

80. Stanford University. *Protégé*. https://protege.stanford.edu/

81. S. Lohmann, S. Negru, F. Haag, T. Ertl. (2016). Visualizing Ontologies with VOWL. *Semantic Web* 7(4): 399-419. http://www.semantic-web-journal.net/system/files/swj1114.pdf

82. SURROUND SoW, *Testbed-17 response by SURROUND Australia Pty Ltd.*

83. TopQuadrant. *TopBraid Composer*. https://www.topquadrant.com/products/topbraid-composer

84. ISO/TC 211 Group for Ontology Management. https://github.com/ISO-TC211/GOM/

85. Sparx Systems, *Enterprise Architect*. https://sparxsystems.com/products/ea/

86. RDFLib pyLODE. https://github.com/RDFLib/pyLODE

87. SURROUND Australia. *Australian/New Zealand 3D Cadastre Standard Sample*.

88. OASIS CIQ v3.0, *Customer Information Quality Specifications Version 3.0: Name (xNL), Address (xAL), Name and Address (xNAL) and Party (xPIL)*. OASIS (2008). https://docs.oasis-open.org/ciq/v3.0/cs02/specs/ciq-specs-v3-cs2.html

89. Sara Brockmans, Sara, Raphael Volz, Andreas Eberhart, Peter Löffler (2004). "Visual Modeling of OWL DL Ontologies Using UML." In *The Semantic Web — ISWC 2004*, edited by Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, 3298:198–213. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, https://doi.org/10.1007/978-3-540-30475-3_15.

90. Knut Jetlund, Erling Onstein, Lizhen Huang (2019). "Adapted Rules for UML Modelling of Geospatial Information for Model-Driven Implementation as OWL Ontologies." *ISPRS International Journal of Geo-Information* 8, no. 9 (August 22, 2019): 365. https://doi.org/10.3390/ijgi8090365.