OGC® DOCUMENT: 21-031

External identifier of this OGC® document: http://www.opengis.net/doc/PER/t17-D023

Open Geospatial Consortium

OGC TESTBED-17: UML MODELING BEST PRACTICE ENGINEERING REPORT

ENGINEERING REPORT

PUBLISHED

Submission Date: 2021-11-19
Approval Date: 2021-12-09
Publication Date: 2022-02-08
Editor: Sam Meek

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

I ABSTRACT

This OGC Best Practice provides readers with guidance on how to use the Unified Modeling Language (UML) within the scope of OGC work. Recently there has been a move to a *resource-based* approach for OGC Application Programming Interface (API) definition through the OpenAPI Specification and away from the *service-based* approach specified in OGC Web Service (OWS) standards. Previously, the interface definitions were almost exclusively XML based, therefore models described using UML class diagrams and conceptual models in general simply mapped 1:1 to derive the XML schema. Using API resources has resulted in the possibility of deriving multiple target technologies from a single standard and therefore, UML model. An additional point of discussion within the OGC is the value added by conceptual modeling using UML. Models included in OGC Standards vary from diagrams only, to conceptual models and model fragments all the way through to Model Driven Architecture (MDA) where UML models are used to directly derive implementable artifacts such as schemas.

UML has been the main modeling language of choice within the OGC, although up until now, there has been little guidance within the OGC on appropriate use of UML. These Best Practices do not seek to govern the use of UML within the OGC as it is recognized that UML is a flexible language that has applications beyond the current OGC doctrine. However, the practices seek to provide guidance to assist in adherence to the following principles:

- Correctness — Adherence to the Object Management Group (OMG) UML standard.
- Consistency — UML artifacts should be consistent across OGC Standards and with supporting standards such as those specified by ISO/TC 211.
- FAIRness — Findable, Accessible, Interoperable and Reusable models.
- Value — Any modeling done, UML or otherwise, should add value to the parent standard. That is, the modeling should do work for the community that is not done elsewhere.

The Practices are as follows:

- Practice 1: UML models should follow the OMG UML 2.5.1 Standard ratified in 2017.
- Practice 2: OGC Conceptual Models should be represented as UML Class diagrams.
- Practice 3: OGC Conceptual Models should be platform independent.
- Practice 4: OGC Conceptual Models should use concepts consistently across standards.
- Practice 5: OGC Standards should contain a UML model at least at the *conceptual* level of detail.

- Practice 6: UML models in OGC Standards should add value.
- Practice 7: UML models should describe structure in the engineering process.
- Practice 8: Modeling artifacts should be provided in full.
- Practice 9: UML models should at least be consistent with supporting text, but ideally normative.
- Practice 10: UML tooling should produce interoperable artifacts.
- Practice 11: UML can be used for modeling semantics, although there are other technologies that are more appropriate.
- Practice 12: OGC UML models should be machine readable (i.e. available in XMI format, in addition to the format of the UML Editor used to create the model).

II KEYWORDS

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

ogcdoc, OGC document, UML, best practice, model driven architecture, MDA

III SECURITY CONSIDERATIONS

No security considerations have been made for this document.

IV SUBMITTING ORGANIZATIONS

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

- Helyx Secure Information Systems

V SUBMITTERS

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

NAME	ORGANIZATION	ROLE
Sam Meek	Helyx Secure Information Systems	Editor

1

SCOPE

1 SCOPE

The scope of this document is UML modeling as part of standards and data models within the OGC standards definition process. This paper is concerned with the UML aspects of conceptual modeling within the development of OGC Standards; these aspects should be governed elsewhere, possibly in other OGC Best Practice documents.

1.1. OGC Standards

The development of OGC standards is the primary driver behind this best practice document. Standards are developed by the OGC community whose membership hails from a variety of backgrounds including industry, academia, defense, and government. The standards definition process is work completed in response to a domain need to standardize data and interactions. Much of the actual work of creating a standard is completed from within the SWGs and DWGs through voluntary contributions from the membership. Additionally, standards are developed somewhat independently of each other; there are governing documents in the form the Abstract Specification Topics that should be referenced during the standards definition process, although this is not always the case.

Defining standards often includes the use of models to describe data and interface structures, as these are the two main aspects to be standardized. The models are usually UML Class Diagrams for describing structure although it is recognized that other modeling languages are more suitable for describing different aspects of a standard. Semantic definitions and relationships are used regularly to provide relationship information between standards, for example, a *feature* in one standard should represent the same information as a *feature* in a different standard. Although the structure of a *feature* can adequately be described using UML, semantic meaning between elements of a standard may better be represented in a language built for semantic description.

The state of existing models within OGC standards varies in quality, scope, approach and utility. This is largely due to the independence of approaches to OGC standard development, but also the variety of challenges that the standards are designed to address.

1.2. UML

Unified Modeling Language (2.5.1) is an Object Management Group (OMG) and later International Organization for Standardization (ISO) standard with history stretching back over 20 years. It was primarily designed to describe the *structure* and the *behavior* of programs, systems, and data models. UML consists of 14 diagrams, half representing structure and half representing behavior with a few of these including *interaction* elements. Although UML is suited to modeling the entire suite of system behavior, the paper is only interested with the

aspects of UML that assist with the OGC's UML modeling aspirations, this is the Class Diagram. Please note that unless explicitly stated, from this point forwards this paper uses the term *UML* to refer to *UML Class diagrams*.

1.2.1. UML Class models

The purpose of conceptual modeling is to describe *structure* rather than behavior or architecture, a suitable and robust approach is to use the UML_Class approach (referred to throughout this document as the *UML Class Diagram*). It is recognized that UML Class diagrams are obviously more than just pictographic representations of structure, when utilized correctly they are full models and a diagram is simply a view on the model.

UML Class diagrams are utilized across the OGC in conceptual modeling and are the go-to approach for carrying out this type of work. This paper is specifically focused on UML Class diagrams and therefore it should be used where UML class diagrams are appropriate in the standards definition process.

UML class diagrams, coupled with the Model Driven Architecture (MDA) process can be utilized for *code stubbing* in Object Oriented (OO) languages such as Java. Although still utilized in niche use cases, class diagrams are not the usual entry point in designing software programs, as there are more efficient ways of doing so. However, class diagrams are useful for:

- Describing objects and their attributes.
- Describing the relationships between the different objects.
- Creating a conceptual view of a structure.
- Deriving lower-level models and schemas.

The remainder of this paper is concerned with a Best Practice for UML class diagrams.

1.2.1.1. Assumptions & Shortcomings

UML as a modeling approach is a defined language and therefore has its own rules and doctrine; the inbuilt assumptions with UML do constrain how the language should be used and the kinds of approaches that are easily represented and ones that require workarounds. An obvious assumption with UML is that it is an Object-Oriented (OO) modeling language where concepts are divided into classes that have attributes and relationships between them of varying type. This approach works well for OO targets such as relational databases and Java classes, other paradigms are modeled using UML, but require best practice to ensure correctness and consistency.

1.2.2. Data modeling

UML Class diagrams lend themselves very well to modeling the structure of programs and by extension, data. Data modeling in the abstract sense can be defined using classes, attributes

with types and even methods for operations. A key aspect of data modeling with UML is that the representation is abstract and not tied to any particular technology. Data models can be specialized into target technologies for physical storage and utilization, however, this is not required for a model to be useful and valid. Data models may also simply be part of a program running and therefore holding data in transit and use as well as at rest.

Although UML Class diagrams for Data Modeling are abstracted, they are inherently OO due to the structure of UML Class diagrams. This does not mean that UML cannot be used to model data of non-OO, but workarounds are required and it may not be appropriate to do so. In terms of best practice, this decision should be up to the relevant SWG, DWG and modeler.

1.2.3. Interface modeling

Modeling an interface definition with UML is much like modeling a data model, except the data is either in transit or in use rather than at rest as a typical data model for a target technology would imply. There is potentially a larger focus on *methods* as well as classes and attributes as interfaces exist to gain access to data or capability in someway. There are some extra considerations for UML for interface modeling, these include:

- Target technology — APIs (whether REST or Services based) that are quite different and may require different models.
- Methods — methods should be described consistently.
- Patterned attributes — interfaces can have patterned attributes such as Paths in REST interfaces, UML prefers fixed attribute names.
- Object Orientation — the corresponding physical artifacts to an interface model are less likely to be OO than data models which requires consideration and workarounds.

The structure of interface definitions in UML are often more complex than data models, especially when a target encoding is JSON. One approach to using UML for model JSON interfaces is to define a new object or class when braces are used. This can result in many objects within objects with a *composition* relationship and can make the visual aspects of the model look cluttered.

1.3. Model levels & terms

When creating models using UML, the way the modeling artifacts are represented depends on several factors:

- Model purpose
- Level of detail

- Value proposition & utility

One such way of organizing types of models is to describe the *abstraction* from the real-world artifact. This hierarchy has crossover with other concepts such as *level of detail*, generalization and purpose, but the driving factor is understanding how closely the model represents an instance of the model. The levels are as follows:

- Conceptual
- Logical
- Physical

These are levels of data model first described in [7].

1.3.1. Conceptual models

Conceptual models are the highest level of abstraction and usually contain the smallest amount of detail. These types of models may come in the form of a UML *package diagram* that just shows the relationships between the different packages or occasionally the external dependencies. Conceptual models may also only show the classes and the relationships between the classes without any lower levels of detail. The purpose of the conceptual model is to organize the important concepts in a domain. For the OGC, the conceptual model should show the major aspects of the model target and also the model's relationship to other models within the OGC and ISO if appropriate.

1.3.2. Logical models

Logical models are a complete, platform independent description of a real-world artifact. The model contains all of the information required to derive physical models, but without the nuances and structures of a defined technology. The model will likely include class attribution, code lists, data types and relationship descriptors such as multiplicity, direction and qualifiers. The key aspect of the logical model is that it is detail complete in terms of its logical components, but platform independent and may still be missing detail required for a specific target technology.

1.3.3. Physical models

The lowest level of detail models are physical models that are by their nature, platform dependent. It is possible that a physical model will not be constructed in UML but may be created out of a target technology such as XSD or JSON schema. Importantly, the physical model has the means to store the data.

1.4. Model Driven Architecture (MDA)

MDA has its own terminology that is similar to the *model levels* mentioned above, these concepts are referenced from the OMG MDA Guide Revision 2.0:

- Platform Independent Model (PIM)
- Platform Specific Model (PSM)

In general, the PIM is closer to the business concepts and requirements, whereas the PSM is closer to the technology.

1.4.1. Platform Independent Model (PIM)

A PIM is similar to either a *conceptual model* or a *logical model* in the above grouping in that it is designed to support the auto-generation of target technologies from a single model. As mentioned previously, this cannot be a complete free-for-all; there has to be some level of logical representation that target technologies have in common in order to represent all of the required concepts. One such example given is the use of relational databases, or Representational State Transfer (REST) APIs or some other group of technologies for which there are many implementations. This creates a logical abstraction layer that requires the following:

- All information should be captured to generate the target technologies is somewhere in the model; this may be in the classes, the relationships or somewhere else.
- The model must have a recognized, representative structure in order to perform a transformation.
- The model must have a consistent, machine-readable output format to be utilized by a target technology.

If one of these points is false, then the transformation is unlikely to be possible, specific to a technology (negating the independence of the model) or at least inefficient. It should be noted that transformation between logical and physical models is done by transformation software and much of the logic to move a PIM to a PSM will be held in the transformation software.

1.4.2. Plaftorm Specific Model (PSM)

The platform specific model can either be the generated model to transform it into technology specific concepts, for example a data type in a PIM might be *String*, the equivalent concept in Postgresql is a *VARCHAR* and the generated data definition language (DDL) that can be ingested specifically by Postgresql requires transformation of these concepts.

Another definition of a PSM is the artifacts required by a target technology that directly result in an implementation, examples of this include:

- DDL for Postgresql
- XML Schema for XML
- JSON Schema for JSON
- JSON definition file and YAML for OpenAPI

The essential difference is that the model is derived to describe concepts in terms of the target technology that should be implementation-ready regardless of the target technology.

1.5. UML model technology agnosticism

Conceptual UML models should capture all of the information required to express a standard in multiple target technologies, this argument can vary in strength depending on the model purpose. Extreme examples of this principle refer to MDA, where the model will be required to capture *a lot* of information to generate the target technologies to a conceptual model that just contains the relationships between internal and external dependencies and packages.

Technology agnosticism is achievable and has many examples in the OGC and elsewhere, however, being completely *approach independent* is more challenging as the fundamental concepts and assumptions may be different. A typical example of this is a relational database, where creating a data model for a relational database approach can be quite easily abstracted from a database technology; the concepts of primary keys, foreign keys, 1st and 3rd normal form are well-known. However, it may not necessarily be possible to represent the same concept in a NoSQL or Graph or triplestore structure as the fundamentals are different.

1.6. Out of scope

There are several topics that are of peripheral interest to the OGC and UML modeling but are not covered as they are outside of the scope of the paper. The out-of-scope elements include:

- Platform or technology dependent requirements — UML is the focus of the paper and technologies such as MDA are mentioned, however, specifics of potential target technologies are not in scope.
- Other modeling languages beyond UML — There are many languages and modeling paradigms that can be used to do conceptual modeling, however, the focus of this paper is UML.
- Best practice and use of Model Driven Architecture (MDA).

- Use of UML to create implementations of overarching standards through class specialization as it is inherently tied to machine readability and the MDA process — the actual details of UML model construction beyond being correct and consistent are not discussed.
- Specific recommendations for existing standards, this type of work should be done by the relevant Domain Working Group (DWG) or Standards Working Group (SWG) using this Best Practice as a reference.

1.7. Best practice criteria

The criteria for best practice is multi-faceted, although it has an overarching motivation to provide *value* to standards implementers. Overall, OGC UML models should fulfill the following criteria, OGC UML models should be;

- Correct in terms of their construction and adherence to OMG Unified Modeling Language 2.5.1.
- Used consistently across the OGC; a representation of a concept or class should be consistent across all of the standards that use it.
- Add value to the standard in which a model sits; this could be as an informative set of diagrams or a normative model.

1.7.1. Correct UML models

Many recent discussions on UML within the OGC (for example this issue) have raised concerns with the *correctness* of UML models, that is, they are incorrect according to the standard or they do not represent the standard sufficiently to match the normative text. This, at best, causes the standard implementer to ignore the UML model, so that it doesn't add any value, or it causes confusion and conflict when the model and the normative text contradict each other.

1.7.2. Consistency

UML models should be consistent across the OGC, that is, when an object or class is represented in one standard then it should appear in other documents in the same way crucially with the same semantic meaning. The best practices outlined in this document aim to provide a framework for consistency of UML modeling across the OGC. For OGC API Standards, one approach to ensuring consistency is to utilize OGC API — Common to formally produce modeling for reusable, high-level concepts. Other suites of OGC Standards may also need similar common conceptual models as has been achieved with CityGML 3, OWS Context, CDB, Symbology Core, and Observations and Measurements (O&M) standards.

1.7.3. Adding value

Perhaps the most important driver of producing the best practices is to ensure that UML models deliver value towards standards initiatives be it through fully implemented MDA or simply providing a conceptual diagram to describe the objects, relationships and peripheral information required to implement the standard outside of any modeling process. Modeling is a time consuming endeavor and requires time investment and skills that are not necessarily held by usual standards creators and implementers, therefore the end product has to deliver value proportional to the investment required to create the model.

The value proposition of a model is dependent upon many things, initially this is model intent. If a standard is structured to be derived using and MDA process, for example, then the value of the model is realized when that process is complete. If the model is created and never transformed, then it may be argued that there is no value in it. However, if the intent of the model is to show diagrammatic relationships between one standard's use of an object and OGC API Common, then there is no need to have it in a state to perform transformations. This criterion simply states that the model creator should understand the purpose of the finished product prior to publication.

2

NORMATIVE REFERENCES

2 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

ISO: ISO 19103:2015, *Geographic information — Conceptual schema language*. International Organization for Standardization, Geneva (2015). https://www.iso.org/standard/56734.html

ISO: ISO 19115-1:2014, *Geographic information — Metadata — Part 1: Fundamentals*. International Organization for Standardization, Geneva (2014). https://www.iso.org/standard/53798.html

ISO: ISO 19157:2013, *Geographic information — Data quality*. International Organization for Standardization, Geneva (2013). https://www.iso.org/standard/32575.html

ISO: ISO 19139:2007, *ISO / TC 211: ISO 19139:2007 Geographic information — Metadata — XML schema implementation (2007)*. ISO (2007).

ISO: ISO/DIS 19115-3, *Geographic information — Metadata — Part 3: XML schema implementation for fundamental concepts*. International Organization for Standardization, Geneva https://www.iso.org/standard/80874.html

OGC: OGC 15-097 OGC Geospatial User Feedback Standard. Conceptual Model (2016)

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

3

TERMS, DEFINITIONS AND ABBREVIATED TERMS

3 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.

3.1. Terms and definitions

3.1.1. **Concept Model**

Implementation independent representation of the nouns that are important for an organization, domain or industry.

Note 1 to entry: A comparison to the term ‘conceptual model’ is presented in Clause 4.4.1 of this Best Practice document.

3.1.2. **Conceptual Model**

model that defines concepts of a universe of discourse (Source: ISO 19101)

Note 1 to entry: For this OGC Best Practice, the term more specifically refers to a representation of a system that consists of concepts used to help people know, understand or simulate a subject the model represents.

3.1.3. Logical Model

A data model of a problem domain that is presented independently of the of a particular database management product or storage technology.

3.1.4. Physical Model

A database or technology specific implementation of the logical model.

3.1.5. Platform Independent Model

A model of a software system or business system that is independent of the specific technological platform used to implement it.

3.1.6. Platform Specific Model

A model of a software system or business system that is linked to a specific technological platform.

3.1.7. Model Driven Architecture

A software design approach for the development of software systems. It provides the guidelines for the structuring of the specifications that are expressed as models.

3.2. Abbreviated terms

PIM	Platform Independent Model
PSM	Platform Specific Model

UML	Unified Modeling Language

4

BACKGROUND

4 BACKGROUND

This section provides grounding work for this OGC Best Practice paper.

UML modeling has been a part of the OGC's standardization process almost since its inception; whether it's conceptual modeling for diagrammatic purposes or logical and physical models in a full MDA pipeline as with CityGML 3.0 and OWS Context. The *success* of modeling efforts has differed from case-to-case, as has the quality, conformance and harmonization of concepts across models.

UML is used within the OGC in general to create data models and interface models in the form of a class diagram. The following sub-sections focus on the UML aspects of the projects specifically, although it is noted that many of the example projects provided more value to the OGC than just their UML modeling aspects.

4.1. UML to GML Pilots

This section provides an overview of the history of the application schema and transformation work completed over the last 20 years. The detailed information can be found in the Engineering Reports for the different initiatives. The full list of documents numbers 18 at time of writing which shows the legacy of this work within the OGC; it was supported by the OGC and Testbed (OWS) initiative from the very beginning. The Testbeds span from the OGC Web Services Testbed-2 (OWS-2) in 2004, all the way to UML-to-GML Application Schema Pilot 2020 (UGAS-2020), which is described in a later section.

The OWS-2 Application Schema Development Discussion Paper provides an introduction to ShapeChange within the OGC with the goal of creating application schemas for NGA data [2]. The initial process is defined to create ISO 19109 Application Schemas in UML with the explicit requirement for them to be machine readable and therefore transformable by the ShapeChange software into GML representations. The process for generating the application schemas was successful, although it was recognized that the time taken to generate the application schemas via the UML process was quite lengthy and it was more efficient to make changes manually or via an XML schema editor if time was precious.

The OGC Web Services (OWS) 3 UGAS Tool Discussion Paper provides an overview of ShapeChange and its installation, use and maintenance [3]. It is the first time in the OGC that ShapeChange is presented as a tool that can provide an Application Schema output.

4.2. UGAS 2020 Pilot

The UML to GML Pilot 2020 (UGAS 2020) was a multifaceted piece of work that sought to derive methodologies for transformation activities enabling interoperability of data exchange

[4]. One aspect of interest in this piece of work was the transformation of UML to JSON schema via an encoding rule. Schema encoding and conversion rules describe how UML models can be converted using the rules into an encoding of choice, the ER provides GeoJSON and JSON as two such examples. Some of the encoding rules described could potentially be generalized beyond JSON type schemas. For example, the Section on inheritance is useful due to the rules suggested, but also because inheritance as a concept is key to UML modeling in general. Another reason to consider inheritance is because JSON schemas do not support inheritance directly and therefore workarounds are described to utilize UML modeling for non-object oriented output schemas. This section reviews some of the pertinent decisions made in the Pilot that have possibilities for transference to the general case and therefore best practices. This section is not a critique on the decisions made during the UGAS Pilot — it is used to provide knowledge transfer into the Best Practices.

To summarize, the inheritance aspects of the UGAS 2020 Pilot include the concepts of generalization and specialization. These two types of inheritance relationships are handled by using the JSON schema keyword "allOf", this *inherits* the properties of the generalized class into the specialized class. This approach begs the question of how types or functions may be overridden, if a use case for that relationship cannot be found. An additional observation is that the practice of specialization offers limited value in the way it is used. This is due to the lack of support for simple sub-typing of complex types (arrays for example). The authors describe a complex methodology for enabling this feature, at least within a single schema, however, they correctly point out the challenge of enabling this. A secondary concern for this approach is the value offered.

The use of enumerations in JSON is good example of a UML → JSON match as both support the concept of enumerations and are used in the same way. Code lists are slightly different in that the representation of a code list in XML, for example, maps well to code lists in UML. However, code lists in JSON schemas are different in that they rely on a linked object to achieve the desired effect; a code list in UML/XML is a list of shorthand codes that have a lookup to a more descriptive version of the code with peripheral, linked information. In the JSON version, the ER describes the use of a linked object, but it is not clear how the code lists capture the information attached to the code.

Basic types are described in UML by using a set of generalization relationships, each type inheriting from its subtype. Although this approach provides the desired output schema, it is a rather verbose way of describing type information. This is a feature of how JSON does its pseudo-inheritance, as described above.

4.3. OGC Testbed-16 OpenAPI

OGC Testbed-16 saw the use of UML Class Diagrams to model interface specifications [1]. The OGC is currently moving away from the WxS (WFS, WMS etc.) suite of standards based on XML and towards the OpenAPI specification which uses JSON and YAML. The OpenAPI specification differs from XML because OpenAPI is an interface standard. UML class diagrams are inherently OO and can be represented in XML documents, therefore compromises are made to ensure that all of the information is captured within the models — capturing all of the information is required

as the UML model was fed into an MDA pipeline and automatically transformed into a JSON representation of an OpenAPI interface.

The approach to utilizing UML was derived using the following principles:

1. Pragmatism over idealism. The project required delivery of implementable artifacts without stipulating a particular approach.

2. Use of specialization to enforce rules within the transformation process. The OpenAPI specification describes rules at the Object level, these objects were realized as classes in the metamodel and rules were written in the transformation software to detect meta-classes and transform accordingly.

3. Appreciation of the transformation tool's capabilities prior to model design. ShapeChange was used within Testbed-17 and one of the considerations is that ShapeChange does not recognize methods in UML classes out of the box. Therefore, all of the OpenAPI object properties were captured as data items regardless of whether that was appropriate.

There are several differences between interfaces and data models, although they are both structured: an interface definition has far more flexibility in how it is realized. If a data model is relational, then it will follow certain rules, such as the inclusion of primary and foreign keys. Contrarily, an interface model has no such constraint and could vary wildly. This has implications for best practice in that, although UML models may be logical and technology independent, they are likely to be approach dependent.

4.4. Conceptual Modeling Discussion Paper

UML modeling within the OGC applies variable approaches. One of the objectives of this Best Practice is to standardize the use of UML across the OGC. There is a discussion paper within OGC (21-041r2) that was written to explore the possibility of a group within OGC to assist with UML modeling in a standardized way across the OGC [5]. The paper re-visited some of the discussions on conceptual modeling within the OGC, the focus was not specifically on UML, but conceptual modeling more widely — it just so happens that the majority of conceptual modeling within the OGC is UML. There are multiple criticisms leveled at some of the conceptual modeling that exists within the OGC, the contents of the paper will not be repeated here. Briefly, the main points included:

- There is an identified need for conceptual modeling within the OGC, although the scope and remit is not clear.
- Conceptual Modeling across the OGC is inconsistent.
- Some standards have missing or incorrect conceptual models.
- Conceptual models are occasionally normative, but usually supportive.

The Discussion Paper provides much of the material that drives the need for this Best Practice and is recommended reading as a companion to this work.

4.4.1. Architecture DWG sub-group for Conceptual Modeling

A conceptual modeling sub-group within the Architecture DWG was created following a vote at the Virtual June 2021 Technical Committee Meeting. The full terms of reference for the group are attached to the Conceptual Modeling discussion paper as an Annex. The purpose of the sub-group is to manage the creation and maintenance of the created models in order to ensure correctness, consistency, FAIRness (Findable, Accessible, Interoperable and Reusable) and value. Although the group stops short of mandating a particular approach, language or type of modeling and has no authority to *govern* models that are produced, it is in the process of creating guidelines for modeling; one aspect under consideration is the appropriate use of modeling languages for applications. At time of writing, the OGC Abstract specification mentions *Conceptual Modeling*, the entry is as follows:

Conceptual Modeling addresses the following questions.

- What are the concepts of interest (things of importance) within the scope of the standard (universe of discourse)?
- What are the relationships between these concepts?
- What information is significant about each concept? What are their attributes or properties?
- What are the related concepts from outside the scope?

This use of the term *Conceptual Modeling* is in contrast to, for example, the use of Concept Modeling that can be found here [6]. This table is included as a reference and update on the discussions around conceptual modeling within the OGC, it is useful background information for determining appropriate use of UML within the OGC. However, as mentioned previously, the semantic work within the OGC is out of scope for this Best Practice and is left here to provide context.

Table 1 — Comparison of a Conceptual Model and a Concept Model (adapted from Ross [6])

CHARACTERISTIC	CONCEPTUAL MODEL	CONCEPT MODEL
What is the core purpose of the concept model?	To initiate and/or orient the engineering of something	To create an inventory of defined words to make statements and to disambiguate those statements
What is the target of the model?	Some capability, which can pertain to inventories, processes, locations, roles, timing, or goals, or some combination thereof	Always concepts and only concepts, which can be about anything
Where are the things referenced in the model?	Often in the real world	Always in people's heads source 'https://rubygems.org'

CHARACTERISTIC	CONCEPTUAL MODEL	CONCEPT MODEL
What form does the model take?	Usually graphical, sometimes with definitions	Always definitions and structural statements, often illustrated for convenience by a graphical diagram
What kind of connections does the model feature?	Structural, sequential, spatial, collaborative, cyclic, or motivational -or some combination thereof- depending on the target of the model	Always semantic — either verbal concepts that can be expressed by verbs or verb phrases) or logical
What kind of engineering does the model address?	Any kind	Only knowledge, at least directly
How deep does the model go?	Usually the first in an ordered trio of models — conceptual, logical, physical — based on level of detail, consideration to design, and conformity to platform conventions	As deep as needed in expressing business knowledge; no counterpart kinds of model for higher-level or lower-level views
How is the model represented?	Usually graphically, sometimes with definitions	Always by vocabulary and definitional criteria, usually illustrated by a diagram

Disentangling the types of modeling (conceptual, logical, physical, etc) is part of the group's work. One important finding established is that UML may not be the most suitable modeling language for modeling concepts. There are other models such as the Web Ontology Language (OWL) that are more suited to other modeling concepts such as ontologies and semantics.

The membership of this special interest sub-group is involved with drafting of this Best Practice document.

4.5. Lessons learned

Representations of different types of output schema (JSON, XML, etc.) lend themselves to modeling in UML to different degrees. One of the more challenging aspects of JSON representations is best described using indentations as an indicator of a new class in a UML model. This was the approach adopted in Testbed-16 and is somewhat reflected in the UGAS 2020 Pilot. In very complex UML models, such as those that represent OpenAPI interfaces, the number of classes and specialization relationships can become very large and difficult to maintain. An example below is one of the *Paths* views from the Testbed-16 OpenAPI thread.

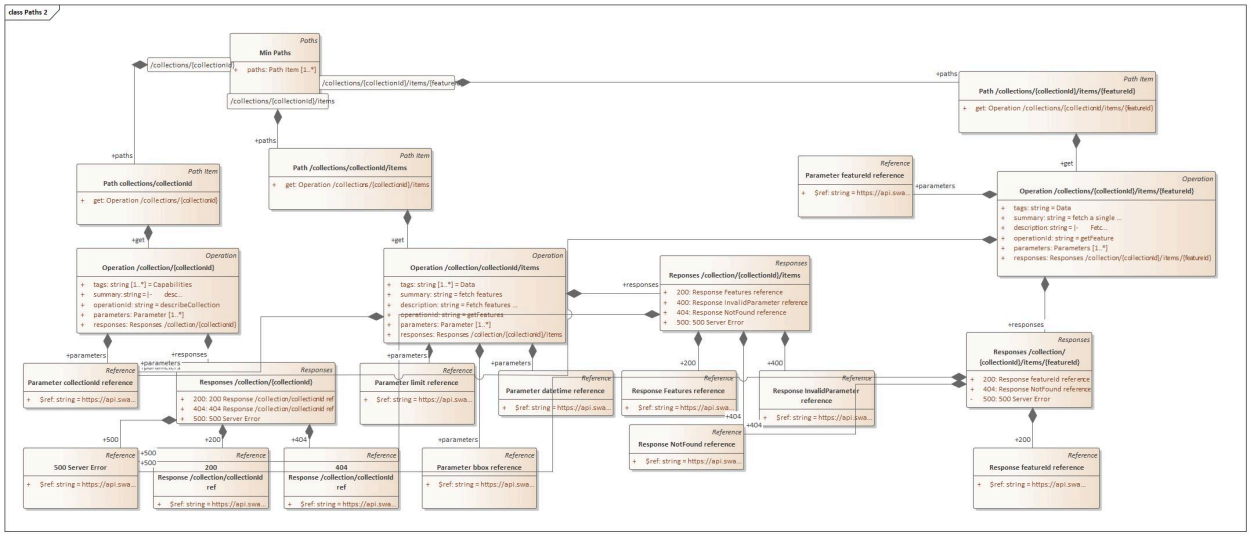

Figure 1 — Paths OpenAPI representation in UML.

The use of UML as the *de facto* language for modeling is not an appropriate assumption. The previous work described in this section, specifically the UGAS 2020 pilot, notes that RDF and SHACL offer better semantic descriptions of relationships. Therefore, the UML best practices should also include some description of relevance of UML as a modeling language and a call for other types of modeling practice.

A lesson learned from the OpenAPI work and the UGAS 2020 work is that adoption of MDA processes requires a lot of upfront work when working with a new encoding, standard or transformation. In the examples given, the MDA process seems to follow the following broad steps:

1. Create a strategy for building a UML model. Translation of domain knowledge into a UML model is not a defined process and a strategy has to be adopted to ensure consistency and correctness. This is not only critical for model readability, but it is also required for simple transformation practices from the UML model into the target technology.

2. Ensure all information is captured within the UML model to an appropriate level of detail. In addition to creating a correct and consistent model, the model must capture all of the information required to derive the target technology output somewhere, otherwise it will have to be injected at another time which increases complexity to the already complex process.

3. Create encoding rules to ensure a consistent and correct transformation. Even with all of the information captured in a consistent and correct way, encoding rules have to be created to define how the model is going to get from its UML representation (described in a Sparx Enterprise Architect Project or XMI) into the target representation.

4. Implement the encoding rules. Implementation is required to actually perform the transformation. In addition to the normal testing required for development

projects, several different models will likely have to be pushed through the transformation technology to ensure that it is implemented correctly and generic.

5. Maintain the models and transformation software. Target technologies are not static and have version uplifts, depending on requirements it is likely that the models and transformation technology will have to be maintained to ensure compliance with target versions of different technologies. Additionally, the process has to be followed for each new transformation required.

Overall, the projects described highlight attempts to utilize UML models and downstream technologies to add value to some or all of the standards definition process. Creating the UML model is not a straightforward process, neither is utilizing the transformation technologies once the UML model has been created.

This section again considers the *value* of UML modeling in different circumstances, it may be that UML modeling is not appropriate in all situations if it offers no value to the overall product. Additionally, the value threshold seems to increase as the complexity of the UML target technologies approach increases; if a UML model maps 1:1 with a target technology, it may be that an MDA process is suitable, as the threshold for implementing it is not very high. However, if a UML model is representing a suite of target technologies over a complex domain, then the value threshold for MDA is a lot higher because the time investment to create it also high.

4.5.1. Modeling status for each standard

The following table outlines the status of each standard with respect to UML modeling. Note that only standards published after 2010 are included and are all taken from the OGC's list of approved standards. Additionally, the table does not provide an in-depth review of each of the models and is there as a survey to understand the proliferation of UML rather than its quality or appropriate use.

Table 2 — The status of various standards with respect to UML modeling

STANDARD	VERSION (LATEST)	YEAR OF PUBLICATION/ REVISION	MODEL INCLUDEI	MODEL TYPE/LEVEL	COMMENTS
3D Tiles specification	1.0	2019	No	N/A	N/A
3D Portrayal Service	1.0	2017	Yes	UML Class/ Package Diagrams	Modeling extensively used throughout the standard
Augmented Reality Markup Language	1.0	2015	Yes	Semantic generic object model	Low-resolution image of model only
Catalogue Services Standard 2.0 Extension Package for ebRIM Profile:	2.0	2010	Yes	UML models used, mixture of	UML package diagrams shown, but the relationships between the packages/classes are unclear and given incorrect

STANDARD	VERSION (LATEST)	YEAR OF PUBLICATION/ REVISION	MODEL INCLUDEI	MODEL TYPE/LEVEL	COMMENTS
Earth Observation Products				relationships and classes	notation. The class diagrams simply represent the tables, value proposition unclear.
Catalogue Service	3.0	2016	Yes	UML models used, mixture of relationships and classes	UML class diagrams used, but missing detail such as multiplicities
CDB (note using Volume 1 core standard)	1.2	2021	Yes	Conceptual	No overarching UML model, snippets are used to illustrate concepts.
CityGML	2.0 (latest in standards catalogue)	2012	Yes	Logical	CityGML is a profile of GML, therefore it uses the GML UML models
EO Dataset Metadata Geo JSON(LD) Encoding Standard	1.0	2020	Yes	Logical	Model included in annexes to describe the profile
GeoPackage	1.3	2021	Yes	Logical, Physical	Extensive UML modeling throughout
GeoSciML	4.1	2017	Yes	Conceptual, logical	Points to WaterML, which contains O&M UML models
GeoSPARQL	1.0	2012	No	N/A	No UML modeling
Geography Markup Language	3.2.2	2016	Yes	Logical	UML models covering Primitives
GeoRSS	1.0	2017	Yes	Logical	UML model in Annex describing GeoRSS Application Schema
GeoXACML	Version 1 Corrigendum	2011	No	N/A	N/A
GeoSpatial User Feedback (GUF) Conceptual Model	1.0	2016	Yes	Conceptual, Logical	Separate document that is specifically there to describe the conceptual modeling aspects
GeoTIFF	1.1.	2019	No	N/A	No modeling anywhere
OGC Water ML 2: Part 4 — GroundWaterML2 (GWML2)	2.2.1	2021	Yes	Logical	GWML2 is Part 4 of WaterML, therefore it reuses a lot of the modeling done in WaterML

STANDARD	VERSION (LATEST)	YEAR OF PUBLICATION/ REVISION	MODEL INCLUDEI	MODEL TYPE/LEVEL	COMMENTS
Hierachical Data Format	5	2019	No	N/A	Contains conceptual modeling, but not UML
Indexed 3d Scene Layer (I3S)	1.1	2020	Yes	Logical	Includes logical schema of certain aspects
Indoor Mapping Data Format	1.0.0	2021	No	N/A	Contains many JSON snippets, but no overarching model
Indoor GML	1.1	2020	Yes	Logical, Physical	Contains logical and physical aspects as it maps 1:1 with XML via GML
KML	2.3	2015	No	N/A	Modeling is described with XML snippets
Land and Infrastructure (Land Infra)	1.0	2016	Yes	Conceptual, Logical	Very extensive UML usage including package and class diagrams
LAS community standard	1.4	2018	No	N/A	No modeling described at all
Moving Features	1.0.2	2019	Yes	Conceptual, Logical	Extensive use of GML, maps 1:1 with implementation
Network Common Data Form (Net CDF)	1.0	2011	Yes	Logical	UML used to communicate data model
Geographic Information — Observations & Measurements	2.0	2013	Yes	Logical	Models included, but are displayed in low-resolution in the standard
OGC-API Feature Part 1:Core	1.0	2019	No	N/A	The standard and bibliography make reference to a UML diagram, but it is not shown in the standard document. Comments show that the UML was removed from the document because it was not considered fit for purpose
Open GeoSMS Standard — Core	1.0	2012	No	N/A	No model included
Open Modelling Interface Standard	2.0	2014	Yes	Logical	Contains a few logical models of elements, although lacks an overall model describing relationships

STANDARD	VERSION (LATEST)	YEAR OF PUBLICATION/ REVISION	MODEL INCLUDEI	MODEL TYPE/LEVEL	COMMENTS
OWS Context Conceptual Model	1.0	2014	Yes	Conceptual, Logical	Full logical model included with reference to other standardized models. Maps 1:1 with XML
OGC Web Services Security	1.0	2019	No	N/A	Contains many XML snippets that could be considered the *physical model*
PipelineML Conceptual Model & Encoding Standard	1.0	2019	Yes	Conceptual, Logical	Contains a lot of UML to describe classes and relationships
Publish/Subscribe Interface Standard	1.0	2016	Yes	Conceptual, Logical	UML models used to describe classes, also includes sequence and timing diagrams
PUCK Protocol Standard	1.4	2012	No	N/A	Modeling Absent
SWE Common Data Model	2.0	2011	Yes	Conceptual, Logical, Physical	Contains the full suite of models including packages, classes and schemas.
SWE Service Model	2.0	2011	Yes	Conceptual, Logical	Extensive UML modeling throughout the standard
SensorML: Model and XML Encoding Standard	2.1	2020	Yes	Logical	Extensive UML modeling throughout the standard
Sensor Observation Service Interface Standard	2.0	2012	Yes	Conceptual, Logical	Includes class and process UML diagrams, very extensive use
Sensor Planning Service Implementation Standard	2.0	2011	Yes	Conceptual, Logical, Physical	Extensive UML modeling throughout the standard
SensorThings API	1.0	2016	Yes	Logical	UML class diagrams describing objects and relationships, but missing multiplicity detail
Semantic Sensor Network	1.0	2017	No	N/A	Standard contains semantic modeling using OWL, UML referenced through
Symbology Core	1.0	2020	Yes	Conceptual	UML class diagrams with no attributes included

STANDARD	VERSION (LATEST)	YEAR OF PUBLICATION/ REVISION	MODEL INCLUDEI	MODEL TYPE/LEVEL	COMMENTS
Geographic Information — Simple Features Access Part 1: Common Architecture	1.2.1	2011	Yes	Conceptual, Logical	UML used to describe overall relationships and detailed classes including attributes
Georeferenced Table Joining Service Implementation Standard	1.0.0	2010	No	N/A	UML mentioned, but not used
Time Ontology in OWL	1.0	2020	No	N/A	Semantic standard, therefore OWL is used
TimeseriesML	1.2	2018	Yes	Logical	Extensive UML models, lots of detail
Two Dimensional Tile Matrix Set	1.0	2019	Yes	Logical	UML diagrams are very detailed, some are missing attribute multiplicities
WaterML	2.0.1	2019	Yes	Logical	UML inherited from Observations and Measurements
Web Coverage Service — Core	2.1	2018	Yes	Logical	Extensive UML modeling
Web Feature Service — With Corrigendum	2.0.2	2014	Yes	Logical	UML model snippets describe Objects and interactions
Web Map Server	1.3	2006	No	N/A	No modeling
Web Map Tile Service	1.0.0	2010	Yes	Logical	Snippets used to describe objects and interactions
Web Processing Service Corrigendum 2	2.0.2	2015	Yes	Conceptual, Logical	UML snippets to describe objects and relationships, UML process diagrams also included
Web Service Common	2.0.0	2010	Yes	Conceptual	UML objects and attributes described in snippets, relationships are conceptual
Geographic Information — Well-known text representation	2.0.6	2019	No	N/A	UML mentioned, but not included anywhere

STANDARD	VERSION (LATEST)	YEAR OF PUBLICATION/ REVISION	MODEL INCLUDEI	MODEL TYPE/LEVEL	COMMENTS
of coordinate reference systems					

4.6. Discussion

Uptake, utilization and quality of UML models is variable across the OGC. Additionally, the concept of *value* is rarely considered explicitly with models sometimes added as an afterthought without a maintenance and governance structure in place. There are examples of models within OGC standards that are normative and therefore the source of truth, should there be a conflict between the model and supporting documentation. There are also models that are utilized as part of an MDA pipeline and schemas as derived directly from the model. However, the most common use of UML models within OGC standards is for illustration of concepts within standards as documents. It is not suggested that this use changes, as MDA can be more of a burden to developers than the supposed value added by following the process. Descriptions of the best practices will therefore not mandate or assume any downstream use of UML models and will consider them to be standalone artifacts that offer value in their own right.

Referencing the Table of Standards in the previous section: many standard definition processes attempt to use modeling within the standard, but this modeling varies from very high-level, conceptual models to low-level encoding models for transformation. There are a few examples of modeling being used to generate multiple outputs from a single model, this is evident in CityGML, which is also used to generate CityJSON, and OWS Context which provide both an XML and JSON encoding. Generating multiple encodings from UML models is utilization of MDA applied to its full potential. However, this is not expected to become a normative process within OGC.

In summary, the OGC utilizes UML modeling for description of concepts, modeling data, and in MDA pipelines. The quality of the modeling varies throughout the OGC and has somewhat fallen out of favor recently with the utilization of JSON encodings that can be created and implemented very quickly.

5

MODELING APPROACHES AND PURPOSE

5 MODELING APPROACHES AND PURPOSE

There are several approaches to modeling and UML is, in fact, a suite of models and diagrams that describes the structure and the behavior of a given system. With interface and data modeling, we are mostly concerned with structure, rather than behavior.

5.1. Modeling Purpose

Consideration should be given to modeling purpose. Modeling is a difficult, complex and often time-consuming job, therefore the following should be understood:

- The value proposition of the work — why is a model being included?
- How the Standard fits into the overall suite of OGC standards, including concept reuse.
- The skills required for the standard development and generation.
- Whether the model is normative in describing the standard.

As mentioned in the previous section, creating models of OpenAPI levels of complexity, as seen in Testbed-16, is a labor-intensive process. Therefore, any modeling work undertaken should have a purpose and *add value* or *do work* that the other aspects of a standard documentation do not. Likewise, it is prudent to streamline the work required of a standard; communication of the relevant aspects of a standard should be described in the text or in the model, but not both: there should be no need to describe it multiple times. Adopting this approach will reduce the likelihood of repetition and conflict between different aspects of a standard document.

Perhaps an outcome of this work is the creation of an OGC model repository to harmonize and maintain all of the models through the OGC in a formal manner.

5.1.1. Communication & Diagram

UML is an effective communication method for describing data and interface structure. This Best Practice is focused almost entirely on UML Class diagrams, therefore this section is concerned with class diagrams used for communication purposes. UML class diagrams used solely for communicative purposes have several advantages, such as:

- Well-defined relationships between classes
- Ability to fix field values
- Stereotypes

- Multiplicity
- Qualifiers

Additionally, UML is a well-known standard with a standardized graphical user interface, making it ideal for diagrams. Another aspect of UML that has been useful for diagramming is the specialization → generalization relationship, which essentially allows the capture of class type and name. This mechanism was utilized in OGC Testbed-16 OpenAPI thread to describe OGC API — Features Part 1: Core in terms of its OpenAPI meta-classes.

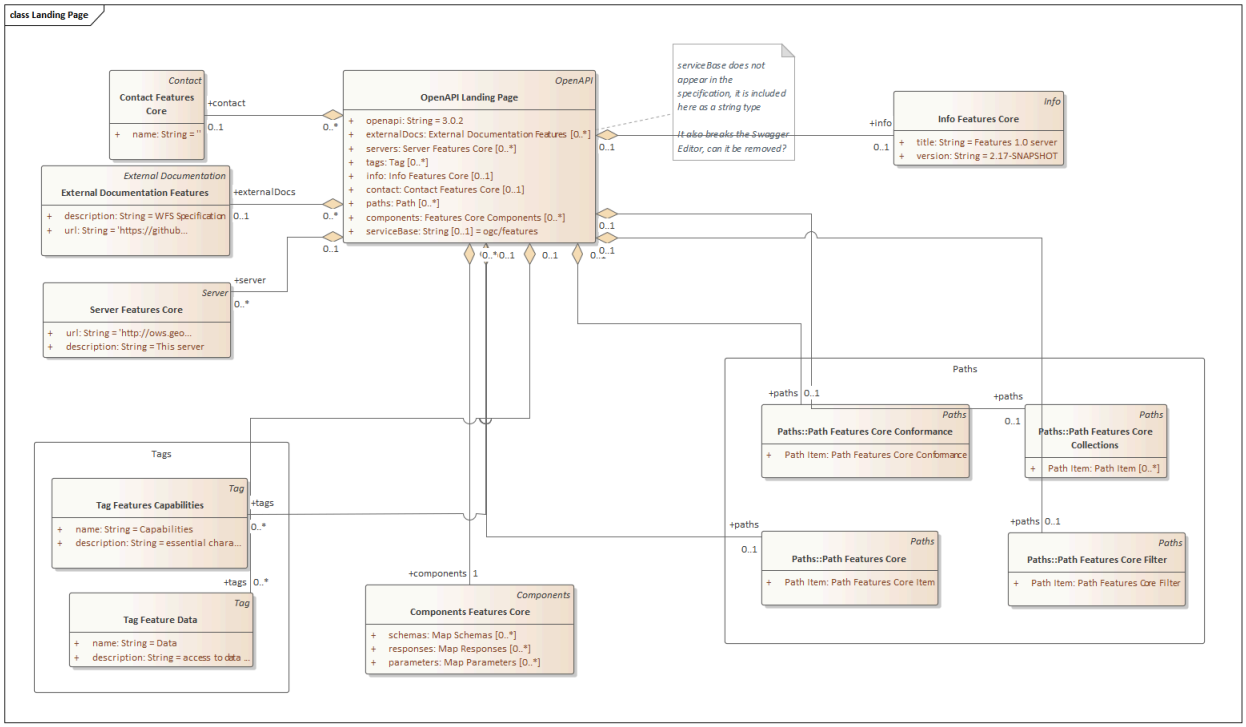

Figure 2 — Use of Specialization Relationship to describe OGC API Class types.

In summary, UML Class models can, as a minimum, be used for creating diagrams to represent a standard. However, if the diagram is complete, correct and consistent, and built in modeling software then it is highly recommended that the model artifacts be supplied for completeness.

5.1.2. Implementable & Machine Readable

There is some confusion about whether a model should be machine readable. Luckily, there are standards governing the transformation of UML models into XMI (XML Metadata Interchange) format, however it is unclear how much this standard is directly used within the community. Transformation technologies such as *ShapeChange* often utilize plugins for modeling software without using the XMI for interchange.

There are several pieces of software that are employed for UML modeling, these include:

- Sparx System Enterprise Architect (COTS)

- OpenMDX (FOSS)
- Dassult CATIA MagicDraw (COTS)
- Papyrus UML (FOSS)
- PlantUML (FOSS)
- LutaML UML (FOSS)

From a best practice perspective, does a UML model *need* to be machine readable? This question has two parts:

1. Does the UML model lend itself to machine readability?
2. Should the UML diagram be provided in a machine readable format?

The structure of the diagram should be consistent across the OGC; if a standard includes a *logical* model with target technologies of JSON and XML, then provided they *are* consistent, they should be machine readable. Likewise, a *conceptual* model that lacks the detail required for transformation may be machine readable, but it may not be appropriate for a machine-readable form to be supplied. The question around *format* is perhaps more important as it requires supply of either an XMI file or a machine-readable version of the UML content (an EAPX file if using Enterprise Architect). Mandating supply of this information for every standard for which a model exists for is likely to be unworkable, however, if a standard is utilizing MDA or an equivalent technology, then it stands to reason that the supporting files should be supplied and maintained along with the rest of the standard.

Before addressing these two questions, it is important to reiterate that UML itself is a modeling language that provides flexibility to data modellers: it provides both a modeling language and a diagramming function, which supports a multitude of diagram types.

The OMG XMI specification allows serialization of both UML models and the UML diagrams in a single XML file. For instance, a UML model can be serialized into the XMI XML format ("UML XMI") while the UML diagrams are provided in serialized form adjacent to that content, in the same XMI file. In fact, this approach makes it possible to only encode machine-readable UML models without any UML diagram.

Technically, with XMI, any UML model and diagram are machine readable since there is a defined mechanism for their serialization into XML.

However, machine readability here takes on a more specific meaning: whether the UML models and diagrams are useful for machines to perform processing.

In the case of implementation, while humans can typically identify and interpret semantic ambiguity, machines require unambiguous information. For instance, an automated, deterministic transform of a logical model to an implementation schema (or code) will require all gaps and holes being filled.

Authors (or data modelers) typically provide UML models that lie between the two ends of a spectrum that describes machine-readability of UML models:

- Strictly modeled UML models that are logically complete — they can be transformed automatically into implementation schemas or software code; or
- vague and ambiguous models that may look natural in diagram form, but are logically incomplete — they do not provide sufficient information for automated transformation into implementation.

Clearly, the former approach lends itself well to machine readability, and hence implementation; while the latter approach treats UML as a diagramming tool rather than as implementation specification.

UML diagrams are useful to humans, but are practically useless for machines to understand in processing a transformation, except for the case where the machine presents those diagrams to humans.

There is an additional complication about compatibility of OMG XMI implementations.

In a perfect world, as long as the data modeler provides a set of logically complete UML models, when encoded in the OMG XMI format, these models should be easily machine readable and implementable.

However, while the OMG XMI specification provides a helpful model-view separation, implementers of UML tools have been way less forgiving to those in the pursuit of data interoperability of UML models.

It is well-known in the UML world that XMI files created using major UML editing tools are not compatible with each other. For instance, XMI files generated using Enterprise Architect are not understood by Papyrus or MagicDraw. This caveat originated from the fact that UML tools typically do more than what is allowed by the UML and XMI specifications.

Most of these XMI generating tools create XMI files that are composed of two parts:

- An OMG XMI-compliant XML part, that contains basic UML models and UML diagrams, encoded under the `http://www.omg.org/spec/UML/20110701` schema's `<Model>` tag; and
- A proprietary XML part that contains proprietary extensions to UML models encoded under the `http://www.omg.org/spec/XMI/20110701` schema's `Extension` tag.

The `Extension` tag contains the encoding of proprietary data related to UML models and diagrams that may or may not be incorporated in the original UML XMI specification, such as:

- description, which can contain rich-text
- stereotype information
- tagged values
- annotations on model elements, such as notes or export format information

- change information, such as last changed date and author information;
- appearance

In some tools, contents included in the OMG XMI-compliant part by default links to the proprietary parts. This means that without an XMI-parser that understands the proprietary information, it is challenging for a machine to understand the full picture presented by the UML model.

For example, in Enterprise Architect, model element description and notes are entered into the proprietary XML portion.

In order to facilitate implementation, machine readable UML models are necessary. The distribution of the modeling source files and their corresponding exported XMI are required in order for automated, deterministic transforms that lead to implementable schemas or implementations. Even though the exported XMI files may not be fully interoperable across tools, they provide a standards-based output that other UML ingestion tools could potentially use for generation of platform-independent artifacts.

5.2. UML Class Diagrams

UML class diagrams remain a popular approach to describing the structure of artifacts. Initially class diagrams were used for code stub generation through MDA, this has somewhat fallen out of favor due to the complexity of the process and the overall utility of the outputs. Although still used for describing structure, class diagrams are more likely to be implemented in data modeling specifically for databases and more recently, modeling of interfaces definitions.

UML class diagrams assume object orientation, although other paradigms are supported with standard interpretation. OO is a popular paradigm for programming and data models; relational databases are implemented in many use cases. However, modern interchange and interface description formats such as JSON and YAML are not object oriented, therefore the design of models with a JSON target require some interpretation. A fuller explanation of utilizing XML to create JSON schemas is in the UGAS 2020 Pilot Engineering Report.

5.2.1. OWL, SKOS & RDF

The graph data model Resource Description Framework (RDF), the modelling language Web Ontology Language (OWL) that uses it, and more specialized systems built on them, such as the Simple Knowledge Organization System (SKOS) are a set of technologies capable of describing all sorts of domains in ontologies. The appropriateness of these technologies will differ depending on the use case in hand. The main focus of these technologies is to produce semantic descriptions of concepts including their relationship to other concepts both in and out of a particular ontology.

The relationship of the Semantic Web technologies to UML and their appropriate use is a topic which is currently under debate within the community. For the purposes of this paper, there is a

distinction between the engineering requirements, best handled by UML, and semantic models of requirements that may best be handled by semantic technologies.

6

TOOLING

6 TOOLING

An aspect of creating models is the tooling available to do so. This section provides a brief overview of some of the tools available to model creators to assist in their craft. As this paper is focused on UML modeling, the tools considered must have a strong UML focus. Additionally, this section does not report on any evaluation work on the tools, it exists to provide the reader with the awareness of particular tools at the point of writing.

6.1. UML diagramming

6.1.1. Sparx Enterprise Architect

Sparx is possibly the most well-known of the UML modeling tools. It provides a full modeling capability and enforces UML rules, although not strictly as some contradictions are allowed. The tool can output the model in XMI and some transformation tools are able to read the native file format. This tool is often used in the MDA process to perform the initial modeling work.

6.1.2. Sparx Cloud

One of the challenges with UML modeling is how to share, disseminate and version control models created by different parties. Sparx Cloud is a piece of software designed to share modeling at an enterprise level. At time of writing, the OGC are in discussion with Sparx regarding a cloud solution for model collaboration. Additionally, the Conceptual Modeling sub-group is concerned with all aspects of modeling within the OGC and are therefore assisting with the possible adoption of Sparx Cloud.

6.1.3. interactive instruments ShapeChange

ShapeChange is an open source, extendable transformation software designed to generate application schemas from UML class diagrams. Its ability to be extended has resulted it being used in multiple OGC Testbeds and Pilots for generating XML, GML and JSON application schemas.

6.1.4. yED

yEd is an open diagramming tool with support for UML class diagrams. Although the tool does not provide an actionable model, it has a standardized, easy to use graphical interface with very little learning required. The downside to the tool is that use of the model is somewhat restricted to the tool.

6.1.5. Microsoft PowerPoint

PowerPoint is often the software of choice for producing presentations. Although it does not have out-of-the-box support for creating and validating UML diagrams, it is often the place where they appear, even as an import from a specialized tool.

6.1.6. Microsoft Visio

The use of Visio for creating UML diagrams can be a point of contention as it is Microsoft's solution for doing so, but does not produce modeling artifacts. It can be used in conjunction with PowerPoint for displaying and presenting UML models, however, it is obviously restricted in its use due to a lack of modeling capabilities.

7

PRINCIPLES

7 PRINCIPLES

This paper runs on four main themes, this section describes each of the themes in turn and makes note of their potential implications.

7.1. Correctness

UML models need to be correct in terms of OMG UML 2017 standard. The implications for producing an incorrect model include the following:

- The model is likely to be misunderstood or not understood at all.
- UML is an abstraction from the things we are trying to represent, therefore misrepresenting it simply introduces further levels of error.
- Having incorrect UML will likely violate many of the other principles

Perhaps a solution to the *correctness* issue is to have the user indicate that their modeling is based upon OMG UML 2017 or if the notation has been partially used to illustrate an object, concept or relationship only. An additional aspect of modeling is that semantics cannot be ignored and may encroach on a model that was originally intended for MDA or other *engineering purposes*. This is to be expected, but the author should at least note whether this is the full explanation, or the model is stored alongside another *semantic* model using the appropriate notation such as OWL.

7.2. Consistency

OGC Standards tend to reference other external documents such as standards from the ISO/TC 211 committee. An additional consideration is that the reuse of concepts, classes and relationships happens across the OGC in different standards, although some of this use should be captured in the Abstract Specification topics. Regardless of their usage or otherwise, one should be able to take an internal or external concept or class and reuse it with impunity — this can only be accomplished if the object is reused consistently across the OGC.

Depending on the requirement or individual model, there is not necessarily a connection to an outside organization or model. Therefore, the dependencies for a UML model should be clearly indicated on the model. There may be a looser relationship between external models and the model in question, these relationships should be documented as well and may be labeled as *inspired by* or an equivalent.

7.3. FAIRness

The FAIR principles are as follows:

- Findable
- Accessible
- Interoperable
- Reusable

Whether models are *findable* or *accessible* will largely depend on how well a model conforms to the first two principles. These principles will also feed into the practice to make models accessible via sharing them along with the other artifacts that come with a standard. ISO TC/211 already adopts this practice and it should also be adopted by the OGC. In the defense of newer standards, there are a lot more UML models available than in older standards so the practice of making models findable and accessible appears to be increasing.

7.4. Value

Value is an ethereal concept with many meanings, *what is value?* In this instance, *value* is about doing work — i.e what work does a UML model do for us? If a model is *normative*, then it can do all of the work, as there is nothing else required to describe a standard. The value added by a UML model should also be proportional to the time and effort required to build it and to use it. There are some immediate values that this practice can offer:

- It avoids recreating something that has already been created — new standards do not have to be defined from scratch.
- Coalescence of multiple standards (or models) is a lot easier if the models conform to the mentioned principles. This adds value to the standards definition process in general as reuse of concepts should ensure a more rapid standard development cycle. One such recent example of this is OGC API — Records, the draft of which was available quickly after inception because it reuses the base structure of OGC API — Features.

8

BEST PRACTICES

8 BEST PRACTICES

8.1. Practice 1: UML models should follow OMG UML 2.5.1 Standard ratified in 2017.

A minimum requirement for UML models within the OGC is that they are correct in terms of the standard. Although obvious, this has not always been observed. This practice is also about fitness for purpose of the UML modeling tool set, for example, UML class diagrams are very good at describing structure, but not designed for describing process, for example.

Another consideration for using UML models is the use of *stereotypes* as there are many standard stereotypes included within the 2.5.1 standard, there are also many in addition to this in ISO/TC 211 that should be re-used where possible. Further discussion on stereotypes can be found in OGC Testbed-17 Model Driven Standards ER.

8.2. Practice 2: OGC Conceptual Models should be represented as UML Class diagrams

It is often taken for granted when talking about conceptual modeling that the term *"UML"* refers to class diagrams, as in the case of this Best Practice paper. UML class diagrams are designed to describe structure, which is suited to the OGC's conceptual modeling use cases.

8.3. Practice 3: OGC Conceptual Models should be platform independent

Going forward, the OGC is supporting different target technologies. As mentioned previously, mapping for model to technology was 1:1, that is, in the web services (or *GetCapabilities*) era, all of the output technologies and descriptions were in XML. However, since the move to OpenAPI and more generally, a resource-based approach, there is the potential for multiple technologies derived or represented from the same model.

There are some assumptions and things to consider regarding this practice. UML Class diagrams are a standard and as such look to enforce rules regarding appropriate use — UML itself is a

language and all descriptions must fit within the language, therefore, *true* platform independence is constrained by the use of a modeling language.

8.4. Practice 4: OGC Conceptual Models should use concepts consistently across standards

UML modeling is not suitable for describing the relationships between concepts in the same way that a semantic language is, however, the concepts in use need to be represented within a UML model in order to make them usable. The semantic descriptions of concepts can be described elsewhere, however, when concepts are used and reused throughout the OGC, they should be done consistently.

8.5. Practice 5: OGC Standards should contain a UML model at least at the conceptual level of detail

OGC Standards should contain a UML model describing the structure of any interfaces or data models, unless doing so violates Practice 6 or is otherwise inappropriate. A model that shows the logical class and relationships between them should be the bare minimum for a standard in order to at least show the use and reuse of top level concepts across the standards. This approach provides a frame of reference for newcomers to the standard as well as allowing the standards designers and implementers to understand *where* any standard fits in with the existing and emerging suite of OGC standards.

8.6. Practice 6: UML models for OGC standard should add value

There are many reasons to create a model for a standard and the appropriateness of a modeling approach will largely depend on the standard in question. Regardless of the containing standard, UML models should *do work* or *add value* that is not otherwise done by artifacts within the documented standard. As mentioned throughout this Best Practice paper, UML models may be used to simply diagram an approach, alternatively they may be used in an MDA pipeline. Both of these approaches, as well as many more, are acceptable reasons to include a model within a standard document.

8.7. Practice 7: UML models should describe structure and in the engineering process

UML modeling is designed for engineering and description of structure, there are other languages such as OWL that are designed to describe concepts. Therefore, we should not be using UML to describe concepts when there are better tools available to us. The OGC approach to modeling in general should be covered in further Best Practice papers. Structure is a key aspect of standards and concepts should be represented in the structure as objects, attributes or methods, the semantic meaning of the concepts should be covered elsewhere.

8.8. Practice 8: Modeling artifacts should be provided in full.

The recent move to using version control seems to enable provision of modeling and other artifacts such as schemas whilst enabling version control and provenance of changes. Although this practice is more prevalent, it is not mandated and modeling artifacts can be lost. Full modeling files and data should be supplied along with the working version of the standard. Following this process is especially important when considering change requests and if the model is normative. In the case of a normative model, any change requests to normative aspects of the document should be led by the model and the supporting text updated accordingly.

Achieving this has been challenging for the OGC, with few discussions on collaboration and version control on models. This is changing through the Conceptual Modeling sub-group of the Architecture DWG and the OGC's potential move to collaboration technologies such as Sparx Cloud.

8.9. Practice 9: UML models should at least be consistent with supporting text, but ideally normative

UML modeling is a powerful tool and done correctly can remove ambiguity associated with the textual and diagrammatic explanations of standards. Additionally, if a model is normative then it is the central location where changes can be made.

8.10. Practice 10: UML modeling tooling should produce interoperable artifacts

There is now a large repository of UML modeling tools both open source and proprietary. The actual choice of modeling software should be up to the modeler, however, the following should be considered:

- Is your chosen tool a *modeling tool* or does it just produce diagrams?
- Does your chosen modeling tool produce an interoperable output, for example XMI 1.0?
- How well does your chosen tool follow the standard? Does it allow you to create illegal classes or relationships and do you have to do *out of bounds* work to ensure compliance?
- If you are looking to implement MDA or some other form of transformation, does your chosen tool have support with a downstream tool? For example, Sparx Enterprise Architect works very well with Interactive Instruments ShapeChange software using the native format. Therefore, the coupling of these tools makes sense.

Overall, this practice is designed to stop vendor lock-in on different tooling and to encourage as much knowledgeable participation as possible through not restricting the tools.

8.11. Practice 11: UML can be used for modeling semantics, although there are other technologies that may be appropriate

There are examples of semantic modeling in UML as relationships can be labeled accordingly to show semantic meaning. ISO/TC 211 requires semantic information to be included in any submitted UML models, therefore there are examples of producing semantic meaning in this way. However, there are tools such as SKOS, RDF and OWL that are designed to do semantic modeling and therefore may be a better choice depending on the use case.

8.12. Practice 12: UML models should be machine readable

The MDA process is not mandated as many SWG leads and other implementers have issues with its efficiency, reliability and overall goals. However, there is a recent push within the OGC, along with the long-running UGAS work to re-look at MDA. The requirements for a model to be suitable for MDA include consistency, correctness and value, however they additionally require the models produced to be machine readable. Therefore machine readability of UML models should be prioritized when creating UML models to ensure interoperability with future endeavors and work with that standard. The models should be available in XMI format, in addition to the format of the UML Editor used to create the model.

9

CONCLUSION

9 CONCLUSION

This engineering report has sought to address the inconsistencies with UML modeling across the OGC. The engineering report has provided a series of Best Practices for UML modeling that are high level enough to be integrated into current practice and low level enough to provide solid guidance. Further work is needed to develop the proposed Best Practices to a stage where they can be applied across future versions of the entire OGC Standards Baseline.

9.1. Future Work

9.1.1. Expand or rectify UML class models contained in existing standards

Currently the OGC has some UML modeling across many of its standards (see standards table). However, some of the models are disconnected from one another. On a positive note, at time of writing, many of the OGC Standards do have at least some UML modeling that could be expanded upon or otherwise rectified to meet the best practice.

9.1.2. Conduct a trial of a shared model repository

There is currently interest in the Architecture DWG for establishment of a shared model repository, similar to that used by TC/211. Future work in testbeds, could include development of procedures and policies for SWGs to use the shared model repository. This could be coupled with the OGC Definitions Server to help manage common concepts.

A

ANNEX A (INFORMATIVE) REVISION HISTORY

A ANNEX A (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	AUTHOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2021-11-19	0.9	S. Meek	All sections	Initial submission

BIBLIOGRAPHY

BIBLIOGRAPHY

1. OGC Testbed-16: OpenAPI Engineering Report (2021)

2. OWS-2 Application Schema Development Discussion Paper (2005)

3. OGC Web Services (OWS) 3 UGAS Tool Discussion Paper (2005)

4. UML-to-GML Application Schema Pilot (UGAS-2020) Engineering Report (2021)

5. Sam Meek: OGC 21-041r2, Conceptual Modeling Discussion Paper. Open Geospatial Consortium (2022). https://docs.ogc.org/dp/21-041r2.html

6. Ronald G. Ross: , “Conceptual Model vs. Concept Model: Not the Same!” Business Rules Journal Vol. 20, No. 1, (Jan. 2019), http://www.brcommunity.com/a2019/b977.html

7. Simsion, G., Witt, G. : Data Modeling Essentials, 3rd edition. San Francisco, CA: Morgan Kaufmann Publishers Inc. (2005)