OGC Testbed-16

OpenAPI Engineering Report

Publication Date: 2021-01-13

Approval Date: 2020-12-15

Submission Date: 2020-11-19

Reference number of this document: OGC 20-033

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D020

Category: OGC Public Engineering Report

Editor: Sam Meek

Title: OGC Testbed-16: OpenAPI Engineering Report

OGC Public Engineering Report

Table of Contents

Chapter 1. Subject

This OGC Testbed 16 Engineering Report (ER) documents the two major aspects of the Testbed 16 OpenAPI Thread. These are:

- A Unified Modeling Language (UML) metamodel that describes OpenAPI and a profile of that model to describe OGC API - Features - Part 1: Core [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html];
- An implementation of a transformation procedure in the ShapeChange [https://shapechange.net/] open source software. This procedure was designed to transform a UML representation of the OGC API - Features - Part 1: Core model into an OpenAPI 3.0 [http://spec.openapis.org/oas/v3.0.3] document.

The process for creating the model and doing the transformation relied upon the Model Driven Architecture [https://www.omg.org/mda/] (MDA) approach. MDA takes a platform independent model (PIM) and transforms that model into a platform specific model (PSM).

Chapter 2. Executive Summary

The OGC Testbed-16 OpenAPI thread and work items were focused on utilizing the Model Driven Architecture (MDA) approach for generating OpenAPI documents. The approach required a Platform Independent Model (PIM) to be transformed into a target Platform Specific Model (PSM) to create the physical artifacts. For Testbed-16 these were a JavaScript Object Notation (JSON) representation of an OpenAPI definition.

OpenAPI is constructed as a specification and has no *official* UML representation. Therefore the UML model describing OpenAPI including relationships, constraints, attributes and classes, was constructed using the specification [https://swagger.io/specification/] as guidance. The OpenAPI specification was modeled in full. However, Application Programming Interface (API) requirements specified in OGC API standards were not explicitly modeled during this Testbed. This approach provides other interested parties with the broadest interpretation of the requirements for an OpenAPI based specification. In the future, business rules may be implemented for OGC API - Common to control the *building blocks* of new OGC APIs. Another approach for future work is to implement OGC API - Common as the metamodel, rather than the full OpenAPI specification.

The modeling aspect of the project required two separate models to be created: The OpenAPI metamodel and a *specialization* of that model to describe an existing OGC Standard. The latter is the UML model that was used in the ShapeChange experimentation. At the time of Testbed-16, the only ratified OGC API Standard was OGC API - Features - Part 1: Core. Therefore, OGC API - Features was used as an example standard to transform using the ShapeChange software.

The approach to using the metamodel-model construct was to invoke a UML specialization relationship. The metamodel provides the generalized versions of each of the OpenAPI constructs created from the specification. This was done to provide the user with rules to describe what should be included as part of any class specialization as well as providing the ShapeChange implementation with a set of supertypes that rules can be implemented against. Therefore, any class specializing from the metamodel is treated the same as part of the ShapeChange implementation. As the OpenAPI specification is not designed in UML, design decisions were made throughout the modeling process as to how to represent relationships and group attributes into classes. This resulted in a highly complex model that fully represents the OGC API - Features Part 1:Core standard. However, *the utility of generating APIs using this approach is questionable*.

The implementation was completed in ShapeChange with a specific goal of creating a pipeline for converting the UML model into a JSON representation of the interface. The approach taken was to create a first pass for converting the UML model into the JSON representation. The lessons learned from the ShapeChange implementation will be used in future Testbeds, potentially involving extra components such as a rules engine.

The recommendations from the project are as follows:

- The project demonstrated that an MDA approach can be used to model APIs using the metamodel/model approach. However, the process is complex and models can quickly become very large. Work should be undertaken to simplify the modeling process. This could be done by creating blocks of modeling artifacts that can be reused when creating new standards and simplify the process for the modeler and model generation process.

- Currently, the only target considered for the modeling work is OpenAPI. A wider consideration for the OGC is whether OpenAPI is the only target worth considering, or whether other targets should be implemented.
- This project raised the question whether there should be a separation between a Standard and an encoding. Currently the OGC API work is heavily invested in OpenAPI, but there will come a time when some other encoding, language or approach may be more suitable. A recommendation is that conceptual models for new standards are the ideal starting point and the encoding aspect is secondary. This observation also highlights the concept of domain knowledge, which is captured in the standard definition process, but expressed in the encoding. Requirements for standards captured in *domain knowledge* does not change very often, but technologies do. By keeping the domain knowledge aspect of a standard separate from the encoding may enable more forward interoperability with future versions of the standard.
- ShapeChange is an excellent piece of software. However, there are some changes that could be made to make the experience of developing with it more palatable. ShapeChange seems reliant on SCXML, which means that any transformations are reliant on the contents of the SCXML. A salient example of this is the association class, where the class information is captured but the relationship between the association class and the relationship it describes is omitted.

2.1. Document contributor contact points

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

Contacts

Name	Organization	Role
Sam Meek	Helyx SIS	Editor
Dan Bala	Helyx SIS	Contributor
Anneley Hadland	Helyx SIS	Contributor

2.2. Foreword

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.

Chapter 3. References

The following normative documents are referenced in this document.

- OGC: OGC 06-121r9, OGC® Web Services Common Standard (2010) [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2]
- OGC: OGC 17-069r3, OGC API - Features - Part 1: Core 1.0 (2019) [http://docs.opengeospatial.org/is/17-069r3/17-069r3.html]
- OGC: OGC API - Common - Part 1: Core (draft) [http://docs.opengeospatial.org/DRAFTS/19-072.html]
- OGC: OGC API - Common - Part 2: Geospatial Data (draft) [http://docs.opengeospatial.org/DRAFTS/20-024.html]
- JSON Schema [https://json-schema.org/understanding-json-schema/reference/combining.html]

Chapter 4. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- **metamodel**

  a metamodel or surrogate model is a model of a model

4.1. Abbreviated terms

- COTS Commercial Off The Shelf
- JSON JavaScript Object Notation
- MDA Model Driven Architecture
- OGC Open Geospatial Consortium
- OO Object-oriented
- SCXML ShapeChange Extensible Markup Language
- UML Unified Modeling language
- XML Extensible Markup Language

Chapter 5. Overview

The rest of this ER is structured accordingly: Chapter 6 introduces the ER and provides an overview of the Testbed-16 OpenAPI thread objectives.

Chapter 7 provides the background to this ER including relevant work in previous Testbeds and Pilots as well as the MDA approach, OpenAPI and ShapeChange.

Chapter 8 discusses the OpenAPI modeling aspect of the project including the metamodel and the derived OGC API - Features model.

Chapter 9 documents the implementation of the ShapeChange extension and process.

Chapter 10 discusses the work and includes recommendations, future work and conclusions for the ER and the project.

Chapter 6. Introduction

This Engineering Report (ER) provides the documentation for the OpenAPI thread in the Open Geospatial Consortium (OGC) Testbed-16 interoperability initiative. The thread consists of two official work items:

1. A UML model and ShapeChange extension to enable transformation of a UML API model to an OpenAPI JSON representation.
2. This document.

The work documented in this ER is a follow-on to the work done in the UML-to-GML Application Schema (UGAS2020) Pilot [http://docs.opengeospatial.org/DRAFTS/20-012.html] (completed November 2020) [1]. UGAS2020 had four main work items, two of which are the foundation for the work described in this Testbed-16 ER. These two work items are: Rules for conversion of UML to JSON schema and initial considerations for using the MDA approach to transforming a UML model of an OpenAPI interface to a JSON representation. The work on JSON schemas is relevant to this ER as the OpenAPI specification uses JSON schema in several places throughout the interface specification.

A UML representation of both the OpenAPI specification and OGC API - Features - Part 1: Core was created. The OpenAPI model is used as a metamodel for creating models of OGC standards. The metamodel is a UML representation of the OpenAPI specification. However, some design decisions were made to fit a JSON representation into an Object Oriented (OO) modeling language. This was done because JSON encodings do not utilize OO concepts such as inheritance. Details regarding this can be found in both the related Testbed-14 ER [2] and the UGAS-2020 Pilot ER [1].

Development of the metamodel was done for two main reasons: 1.) Understanding the process for modeling the specification in UML, and 2.) Creating a model to specialize implementations from. As the implementations are specialized, it is possible to enforce the conversion to JSON in ShapeChange using rules defined at the metamodel level. For example, the OGC definition of a Landing Page is a specialized version of OpenAPI [https://swagger.io/specification/#openapi] type defined in the specification.

OGC API - Features - Part 1: Core was modeled via specialization of the metamodel and using ShapeChange to produce a JSON representation of the API definition. This approach was chosen because at the time of Testbed-16, Part 1 of the Features API was the only OGC standard from the OGC APIs suite approved as an official OGC Standard. The Features API also contains many of the building blocks specified in the draft OGC API - Common. A drawback to the approach of modeling an OGC standard is that it is complex in MDA terms and utilizes much of the OGC API specification. In retrospect, first attempting using the MDA process on a simple OpenAPI implementation and at a later date attempting complex models would have been prudent.

Chapter 7. Background

OGC has adopted OpenAPI as the basis for its *resource* based suite of standards that are being developed in parallel with the XML based web services standards. The first OGC standard using OpenAPI was OGC API - Features. This new OGC Web API complements the OGC Web Feature Service (WFS) standard and may be used as a replacement. Alternatively, a facade between the OGC API -Features instance and the OGC WFS endpoint can be implemented.

OpenAPI utilizes REST endpoints and well-known HTTP verbs including:

- GET
- POST
- PUT
- OPTIONS
- HEAD
- DELETE

The development of OGC Web APIs to JSON based OGC APIs is a major shift in the work of the OGC and has ramifications beyond simply defining and standardizing REST endpoints. For example, many of the return types for geospatial data are XML based, such as GML, as well as the ISO standards that form the basis of defining "returns" from metadata catalogs. Therefore, moving to an API based structure involves having sensible return types and schemas while maintaining the domain knowledge of the standard.

In addition to OGC API - Features, there are other emerging standards being developed to complement the OWS approach. There is also a foundation API standard named OGC API – Common that is being developed. At the time Testbed 16 was being executed, the following new OGC Web APIs were in development:

- OGC API – Common – Part 1: Core
- OGC API – Features – Part 2: Core (approved on November 2nd, 2020)
- OGC API – Coverages – Part 1: Core
- OGC API – Records – Part 1: Core
- OGC API – Processes – Part 1: Core
- OGC API – Tiles – Part 1: Core
- OGC API – Maps – Part 1: Core
- OGC API – Styles – Part 1: Core
- OGC API – Environmental Data Retrieval

The draft OGC API - Common contains common requirements across each of the emerging Web API standards. This standard was particularly important for this Testbed 16 activity as it helps define the rules that go beyond what is supported in the OpenAPI specification. The specification of standards within the OGC is performed in Standards Working Groups (SWGs), with input from the

Domain Working Groups (DWGs). These groups are responsible for generating new standards or contributing to existing standards based on a particular domain of interest. The process of contributing to domains or standards varies across the different working groups and therefore the process of creating new standards is also different across the groups. Some groups start with an existing standard and adapt it based on requirements defined from use cases, others are created from first principles such as in an OGC Innovation Program initiative, and some are brought into the OGC from external communities.

Using the Model Driven Architecture (MDA) process is one approach to attempt to standardize generation of artifacts from conceptual and logical principles. MDA has been mainly used to generate data models for specific platforms from a common logical base. An oft-used strategy is to create a logical model, usually in the form of a UML Class diagram, and then put that model through a piece of software to generate a target output. An advantage to this approach is that the logical model is created once and maintained in a single place and the updated artifacts can be generated automatically for each physical model or implementation. However, the modeling can be complex and therefore unwieldy for all but a few experts with knowledge of the approach and process. Additionally, a typical MDA process starts with a UML class diagram.

This makes the assumption that the platform independent model is object-oriented (OO). In principle, the MDA approach can be used to save time and resources. However, MDA does require development upfront to generate the artifacts for a particular platform. Additionally, if there are changes to the target technology the code to generate the artifact for that platform may also have to be modified. One of the bottlenecks in MDA is the availability of code to generate suitable targets. Additionally, there are usually exceptions to generic rules in MDA. Therefore, targets have to be modified before they can be used.

7.1. Previous work

Using UML to model data structures and interface definitions is the original purpose of the MDA approach to architectures. This section does not provide a history of the use of MDA in geospatial technologies as this ER assumes that the reader is familiar with MDA.

This section presents the recent work done within OGC Testbeds and Pilots to provide a grounding for the decisions made in the OpenAPI Thread of Testbed-16.

The UML to GML Application Schema (UGAS) Pilot is the work that directly precedes the Testbed-16 OpenAPI activity. The output of that Pilot was an Engineering Report (https://portal.ogc.org/files/?artifact_id=95469&version=1) and a version of ShapeChange that was used in the OpenAPI thread of Testbed-16.

The UGAS Pilot sought to address four key issues with respect to JSON schemas:

1. Refine the UML to JSON rules for the NSG wide UML application schema first established in OWS-9.
2. Create JSON encodings for relevant conceptual schemas such as the ISO 19100 series of standards.
3. Understand the utility of the Shapes Constraint Language (SHACL) for Resource Description Framework (RDF) documents and the transformation process from UML to JSON.

4. A preliminary investigation into the utility of deriving OpenAPI 3.1 conformant JSON schemas using the MDA approach.

Points 1 and 4 are all relevant to the work done in Testbed-16 and are reviewed in the following paragraphs.

The UML application schema to JSON schema rules are primarily defined to convert application schemas, that is, a data model for a particular application to a JSON representation. The conversion rules are designed for a direct mapping between UML classes to JSON fragments without implementing inheritance in an object-oriented manner. The authors correctly point out that JSON schemas do not directly support the concept of inheritance, which can be managed through a virtual inheritance approach described in the ER. Additionally, as rules are enforced at the supertype level, but not the subtype level, JSON schema does not support class specialization. The approach to class specialization used in this (or UGAS?) was to extend a class data type to and include both in the JSON schema as both are part of the model. A different approach taken is to use a metamodel, model construct where the metamodel is abstract and the model and classes specialized from the generalized metamodel. This ER describes the completed initial work into deriving OpenAPI based interfaces from UML and identifies the building blocks that are required to produce a successful UML model. However, the participants in the UGAS activity did not attempt to produce a UML model to define the interface due to that work item being included in Testbed-16.

In Testbed 14, the participants in the application schema-based ontology development work activity attempted to express Web Ontology Language (OWL) in UML that could then be translated via an MDA process and ShapeChange into an Application Schema. This is a similar problem to the OpenAPI work described in this Testbed 16 ER but with a different target based upon different encoding rules. Essentially, the participants had similar challenges with how to express out of bounds information, much in the same way that conformance classes are used in OpenAPI to describe conformance to clients.

The Testbed-14 participants also wrote an ER that describes conversion of UML application schemas to JSON and JSON-LD (http://docs.opengeospatial.org/per/18-091r2.html). The authors identified several issues with conversion of application schemas to JSON that included: Lack of support of typical object-oriented concepts in JSON such as inheritance, lack of methods to verify schemas in JSON - notably the lack of the ability within JSON to identify its parent schema, and lack of support for other validation mechanisms such as namespaces. The MDA approach requires many of these concepts to either be directly available or to be virtualized. Therefore, many design decisions made in the modeling process were made to make up for these short comings.

One of the recommendations of this ER was to develop an extension to ShapeChange with the JSON schema target. Although this project was not designed to realize that goal, it did provide some lessons learned for would-be implementers that are discussed in a future section. The second recommendation was to develop JSON schemas for ISO schemas. This is a broader question that needs to be better understood. ISO separates their conceptual model standards from their encoding standards (ISO 19115-1:2014 and ISO/TS 19115-3:2016). JSON potentially needs the same treatment, which would have real-world implications for emerging OGC standards such as OGC API - Records as that in development API will serve metadata that is largely ISO derived.

OGC API – Features was the first OGC Web API Standard to be approved that uses OpenAPI to document the API based on the resource paradigm. As OGC API - Features was the first Web API to be approved by the OGC membership, it provides a template for other, new OGC API standards. OGC API - Features has several parts. At the time of Testbed 16, only OGC API - Features - Part 1 was approved. Therefore, Part 1 was considered in this Testbed 16 project as a case study. The structure of the Features API provides a foundation for other standards in the emerging OGC API suite. For example, OGC API - Records has the same structure as OGC API - Features but instead contains conformance classes such as FullTextQuery that allows for operations such as searching metadata records. An additional reason for using API Features is that the draft OGC API - Common uses many of the conformance classes in Features. Therefore, these classes will likely form the basis of the building blocks for future work in the OGC API suite of standards.

7.2. OpenAPI

The OpenAPI Specification (formerly Swagger Specification) is an API description format for APIs that implement principles of Representational State Transfer (REST). An OpenAPI file allows the developer to describe an entire API including: Available endpoints (/users) and operations on each endpoint (GET /users, POST /users), Operation parameters, and input and output for each operation, and so forth. The use of OpenAPI was experimented with during the development of OGC API Features. Based on that experience, the OGC Membership approved recommending the use of OpenAPI to document any OGC API standard either in development or in the future. OpenAPI provides a well-defined specification for documenting a structure and conformance rules to be adhered to when implementing an API. Additionally, there are many developer tools for generating an API document or stubbing code from an existing API document.

However, OpenAPI is not object-oriented and therefore modeling the OpenAPI specification within the UML class construct requires compensating for this issue. Many of the relationships between OpenAPI objects are straight forward, and can be represented using a simple composition or aggregation depending on the relationship. However, OpenAPI does contain more difficult constructs such as map types (aka dictionary - https://swagger.io/docs/specification/data-models/dictionaries/) in which key/value pairs are both variable. This is slightly more difficult to model in a UML class diagram that relies more upon simple attribute names and variable of fixed type. Additionally, OpenAPI contains arrays of types that need to be ordered, which also are difficult to represent.

Another feature of OpenAPI is that it inherits and extends JSON Schema in an object-oriented manner, but does not explicitly define JSON schema types in the specification. Therefore, a true UML model would also include all of the aspects of JSON schema and all of its extensions. While possible, this was considered out of scope in Testbed 16.

7.3. Model Driven Architecture (MDA)

MDA has been a recognized approach to artifact design and generation for decades. One of the motivating principles behind MDA is the approach to design and the ability to remain technology agnostic. MDA involves creating an initial model, called the Platform Independent Model (PIM) and then generating Platform Specific Models (PSM) automatically from the model. This provides several advantages including: Standardized documentation of artifacts, ability to change implemented technologies without changing the underlying model and the ability to extend the underlying model and quickly generate the artifacts required for implementation.

Generation of OGC API standards using the MDA approach is sensible as the new suite of OGC API standards utilize the same underlying model. Changes and extensions to the model are allowed. However, this can be done in a consistent and therefore controlled manner. The modeling language for the MDA approach is often UML class diagrams as they describe the class attributes, relationships and methods. This combination of entities allows the model to express both structure and behavior and is therefore suited to modeling APIs and associated data schemas.

MDA can also be used for interface definitions. However, MDA has also been used for data structure definitions using UML class diagrams. When employed for data definitions, the transformations are rather simplistic as it involves taking the PIM and transforming it into different versions of SQL rather than enforcing encoding rules used for an interface definition.

7.4. ShapeChange

To transform a UML representation of a model into a configuration file for a target platform, software is used to perform the translation and transformation. ShapeChange [https://shapechange.net/] was the choice of software for this job in Testbed-16 as it has been utilized in previous Testbeds and Pilots and is mature enough to enable a lot of the functionality required. ShapeChange supports JSON schemas through the work completed in UGAS2020 Pilot and the version used in this project was the development branch taken from the recently concluded UML to GML Pilot as this Testbed thread essentially picks up where that piece of work left off.

ShapeChange connects directly to an Enterprise Architect (EA) model created using the EA (https://sparxsystems.com/) software. Through some configuration, it then takes the UML model and transforms it into ShapeChange XML (SCXML). The SCXML is the *starting point* for the development effort to convert the UML model into the chosen target, that is, the developer works only on the XML and manipulates it into the format required for the target. The target is then output from ShapeChange according to the rules applied to the SCXML.

Chapter 8. OpenAPI Modeling

The OpenAPI component delivery has two main aspects to it: The UML model and the ShapeChange implementation. This section covers these aspects in turn with an initial focus on the design decisions made for the UML modeling exercise, and the ShapeChange implementation.

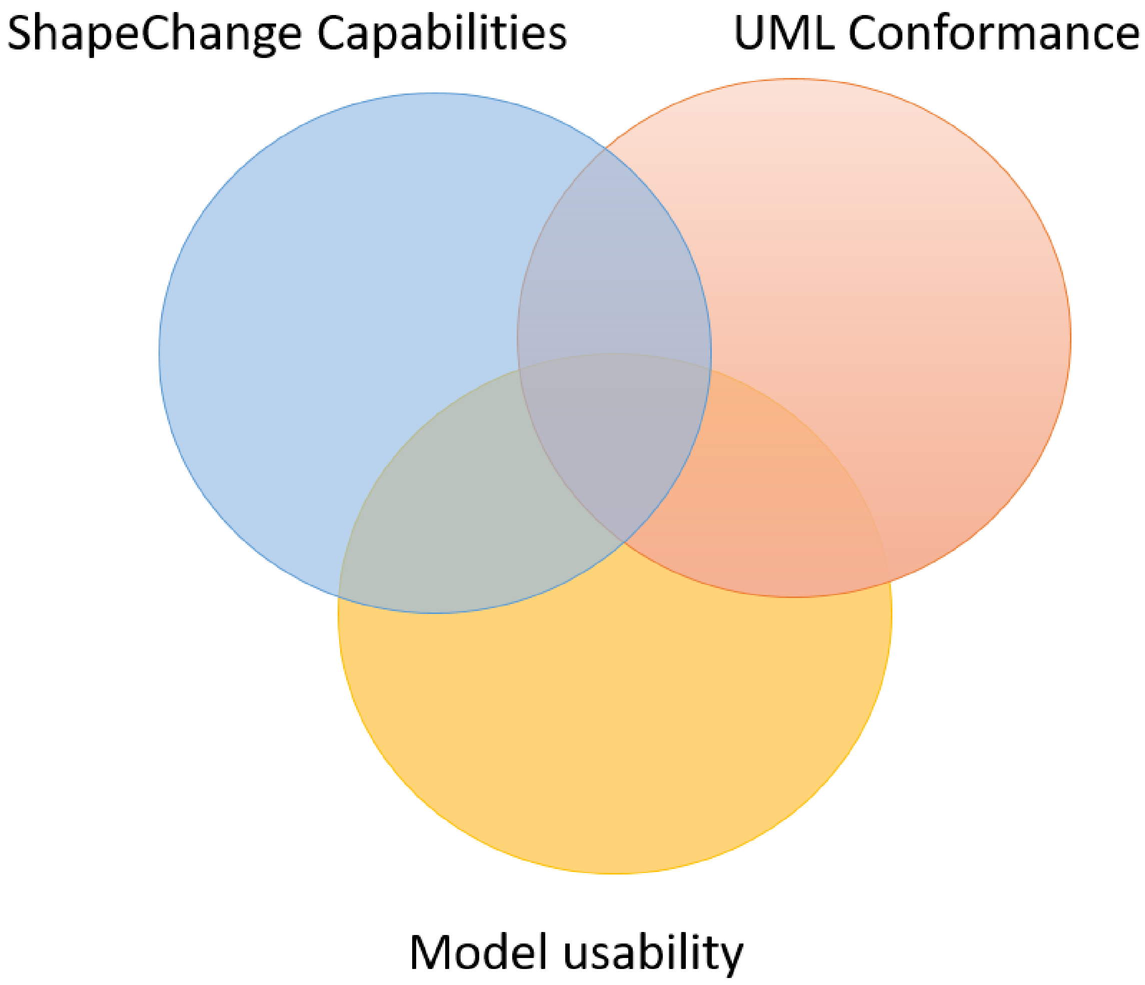


Figure 1. The OpenAPI Main View

Overall, the design decisions were made to manage the trade-off between certain aspects of the project. These are as follows:

- UML conformance - UML models are governed by an ISO standard and the usage of the different aspects of the modeling language is defined and controlled by said standard. Therefore, UML conformance is a requirement of the model, else it is not understandable by would-be users or ShapeChange.
- Model usability - The model described was created with usability in mind, this includes aspects such as; consistency, readability, traceability and reuse of elements. This aspect was potentially the most challenging, as the OpenAPI specification is complex at times and geared towards JSON encodings that do not support typical OO constructs such as inheritance.
- ShapeChange capabilities - UML models ingested in ShapeChange are converted to SCXML prior to being *worked on* by any implemented code and extension to ShapeChange. Therefore, implementing an extension to ShapeChange uses the SCXML representation as a baseline and

any further implementation to ShapeChange's capabilities starts with the SCXML representation of the UML model. An example of a shortcoming in ShapeChange is that whilst is recognizes *association classes* as entities, it does not recognize the connection that is made between the association class and the relationship it refers to. Therefore, any use of an association class is a challenge because SCXML does not specify the relationship it refers to.

- Implementation time - OGC Testbeds are designed to understand ideas and produce proof of concept software as well as making recommendations for future work. The Minimum Viable Product (MVP) for this work is to create a UML model and ShapeChange implementation to produce a JSON encoding and representation of an OpenAPI interface. There are several out-of-scope items that include:
  - Validating models (beyond the model-metamodel relationship).
  - Enforcing OGC conventions such as conformance classes.
  - Producing a generic solution that covers all eventualities.

8.1. OpenAPI UML Models

As mentioned in the introduction, there is no authoritative UML model for OpenAPI that was suitable for the needs of the project. Therefore, a UML model was created from the OpenAPI documentation. Although the model represents the OpenAPI standard and guidance in the most accurate way possible, there is a trade-off between the following factors:

- UML 'correctness' - the model is built in UML and should be compliant with the standard.
- ShapeChange requirements and access to internal variables - ShapeChange needs access to the internal variables expressed in the UML model in order to implement them.
- Aesthetic quality - the model needs to be logical and readable to encourage use.

The UML modeling described and completed in this thread is two-fold:

- An OpenAPI metamodel created from the specification using the relationships and classes described.
- a model that implements the metamodel, OGC API - Features - Part 1 Core.

This approach was chosen to provide future implementers with a metamodel that they could use as a reference for further OGC standard development and documentation.

8.2. OpenAPI UML metamodel

The OpenAPI metamodel was created from the OpenAPI specification [https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md] version 3.0.2 and is presented as a set of classes that interact via relationships. OpenAPI is prescriptive about the majority of the OpenAPI objects and relationships. However it is not strictly OO. Therefore, interpretations and design decisions were made to represent OpenAPI as closely as possible whilst adhering to UML convention. An additional factor that influences the design of the model was the requirement to implement a plugin to ShapeChange to produce the OpenAPI document from the UML model using an MDA approach. This has a direct influence on the use of *methods* within UML, as ShapeChange currently does not recognize *methods*, therefore the *operation* call within OpenAPI would naturally sit as a *method* in a class diagram, but has been included as an attribute due to the shortcomings of ShapeChange. This section shows the views of the UML model as well as some of the design decisions taken.

The metamodel was designed with consistency as a key factor for determining success. For example, the Reference object could be presented as a String type with the name $ref and the value as the variable. From a ShapeChange implementation perspective, this would be the simplest option, as it would follow the typical key/value pairing. However, many reference objects are reused throughout the model and have rules enforced upon them. Therefore, from a UML perspective, implementing a Reference class is prudent so that the specializations of the Reference class can be acted upon accordingly. A similar approach was taken for the Map class, which is discussed later in this section.

The metamodel design process included the entire OpenAPI specification and was done as a reference for implementing classes and as a hook for the ShapeChange implementation. The OGC API - Features - Part 1: Core specialization only uses a subset of the metamodel, but the full model is available in the accompanying Enterprise Architect project.

8.2.1. OpenAPI Main view

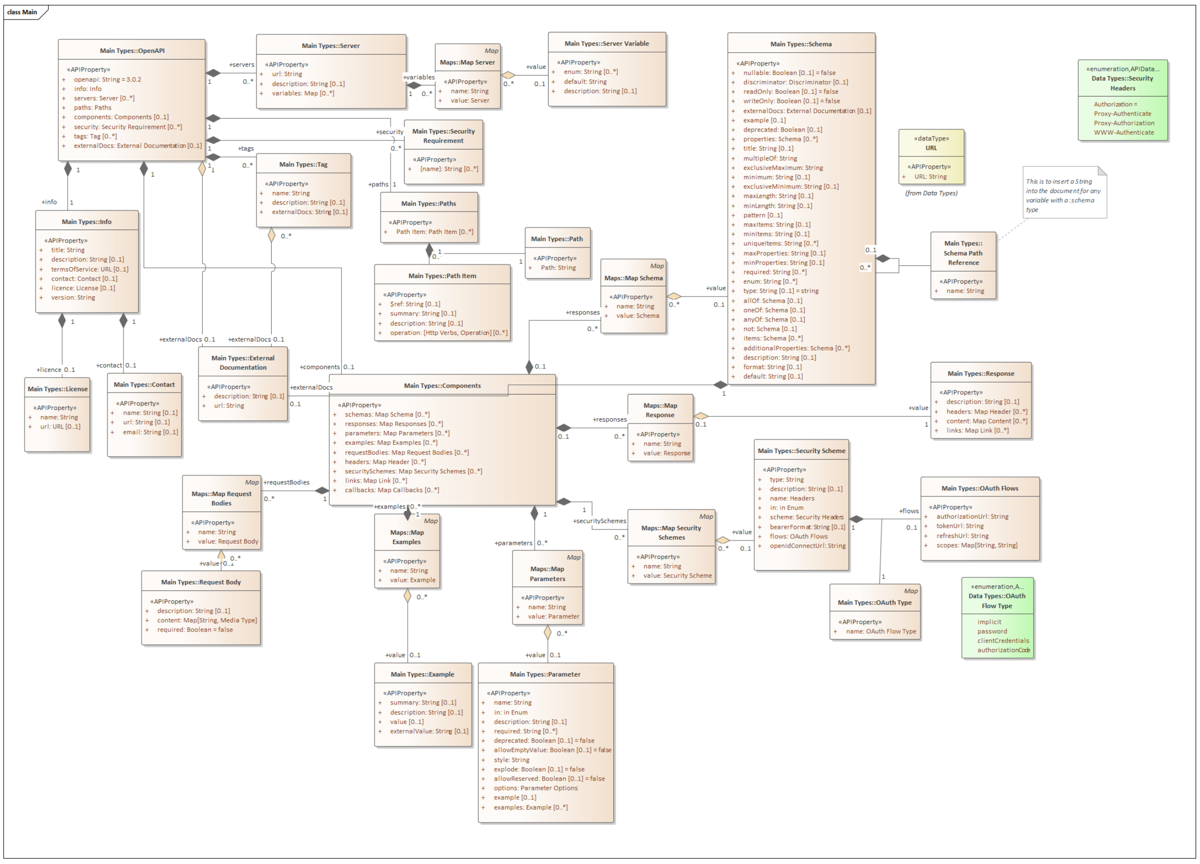


Figure 2. The OpenAPI Main View

The OpenAPI main view in Figure 2 shows the OpenAPI class as an entry point for the specification, it sits at the top level of the resultant OpenAPI document tree. The model follows the OpenAPI specification with relationships between the classes as *compositions* as the *conceptual* relationship between the classes is strong; in JSON when encapsulating brackets are destroyed, the inner contents of those brackets is also destroyed. This maybe a heuristic for modeling JSON in UML generally. The OpenAPI type is also straightforward in terms of its variables with each having a simplistic key, value or key and array of values paring.

8.2.2. OpenAPI Paths views

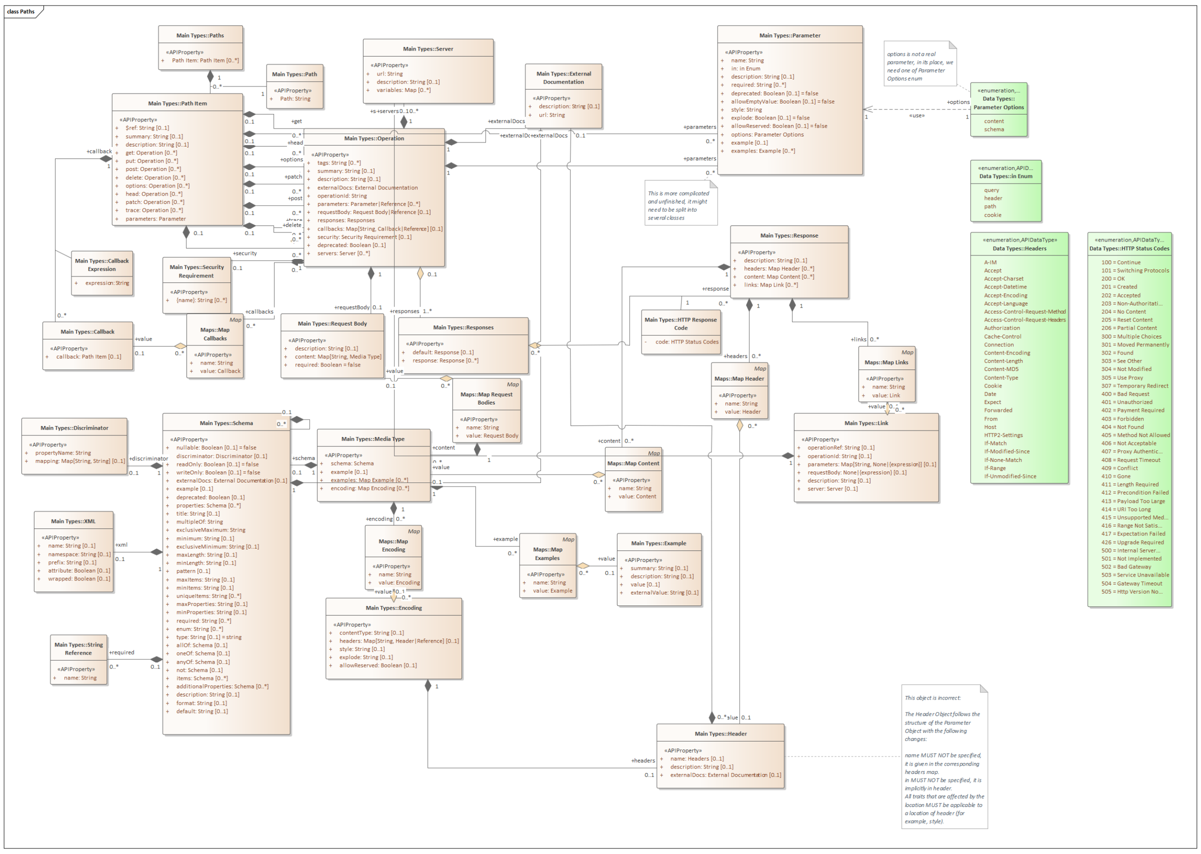


Figure 3. The OpenAPI Paths View

The Paths view is an extension of the main view but separated out for readability. As mentioned previously, the paths class is where the path of a resource is located. These paths lend themselves to be UML operations. However, ShapeChange does not process operations and the path would therefore not appear in the output SCXML. Therefore, operations are described as attributes with the value representing the operation body and an association class used to inject the path variable.

The types of operations that can be performed are restricted to the HTTP verbs and marked as optional. If operations were represented as *methods* in the UML diagram, then restricting the vocabulary of an operation would be simple but still making it mandatory so that at least one exists. This was more difficult to show using the attribute construct as it requires attributes to have knowledge of each other. Another way to represent it is to have an operation attribute with a restricted vocabulary or enumeration as the value and the multiplicity of 1..*. This would ensure that an operation exists. However, more work would have to have been done in ShapeChange to correctly format any operation variable within a Path class.

8.2.3. OpenAPI Maps views

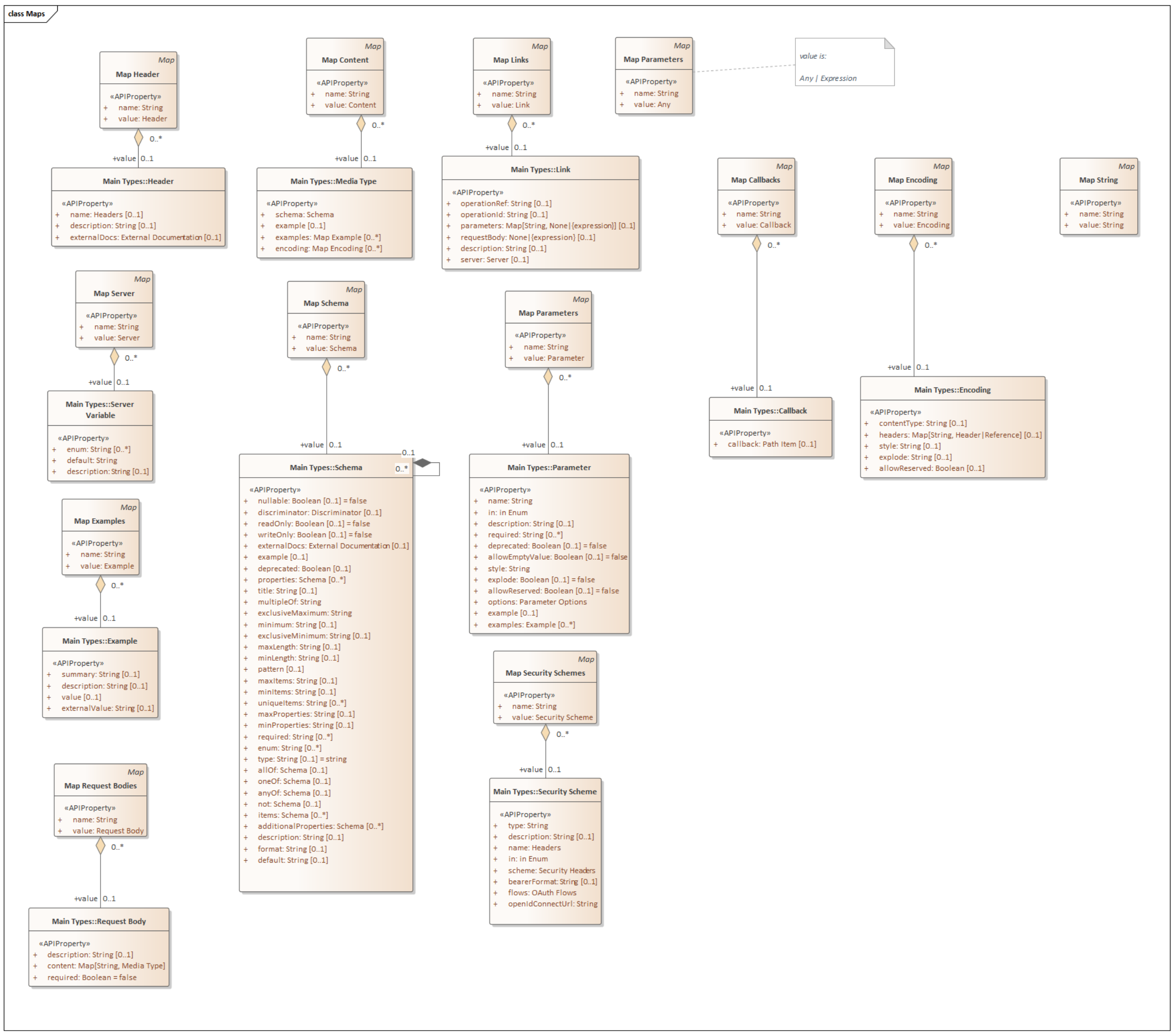


Figure 4. The OpenAPI Maps View

The OpenAPI standard contains a Map type, which is a typical key, value pair_. However, the map is specialized according to the value type is can contain, for example, a Map Object that maps responses to the Components class has a String, Response Object or Reference as its acceptable types. These Maps were specialized to control the data type allowed for each Map type, for example, a *Response Map* contains a name (the response code) and a value that must be of type *Response*. Another method to represent this is to have a single Map type and enforce the value type arguments at another point in the process such as within ShapeChange or via a separate process.

Another issue with the Map type and other types across the specification was control for variable types in attribute values. For example, all Map types have a defined class at their value, *or* a Reference type. In UML, this could have been done by simply including a pipe (|) in the attribute value. However, this is not strictly allowed in UML *and* the ShapeChange implementation would have to control for pipes in the attribute values to enforce the associated type. Map types with multiple attribute value types were controlled using a double specialization to show extension of a Map type with an Object type as its value.

8.2.4. Association classes

Use of association classes has been minimized as Map types usually account for a name and a value. However, there are certain instances within the OpenAPI specification when this is not the case. The main examples are in the Paths Object and the Schema object, the Callbacks object also contains an association class. The use of the association class was deemed suitable as it has the same meaning whenever it is used. From a UML persepctive this happens when the OpenAPI specification records the attribute name as a variable type and the value is an encapsulated class often with a controlled vocabulary. For example, the response type in the paths object records the following:

- The response code as the attribute name.
- The instance of the response class or reference as the attribute value.

This translates in UML to:

- *Default* is the default response as an attribute name.
- The type is *Response* to indicate that it a response type.
- The association class consists of a variable called *code* controlled by an HTTP response code as an enumeration.

This approach has shortcomings in that rules need to be encoded in ShapeChange to interpret the variables in the association class to produce the correct output document.

8.2.4.1. Association in Paths objects

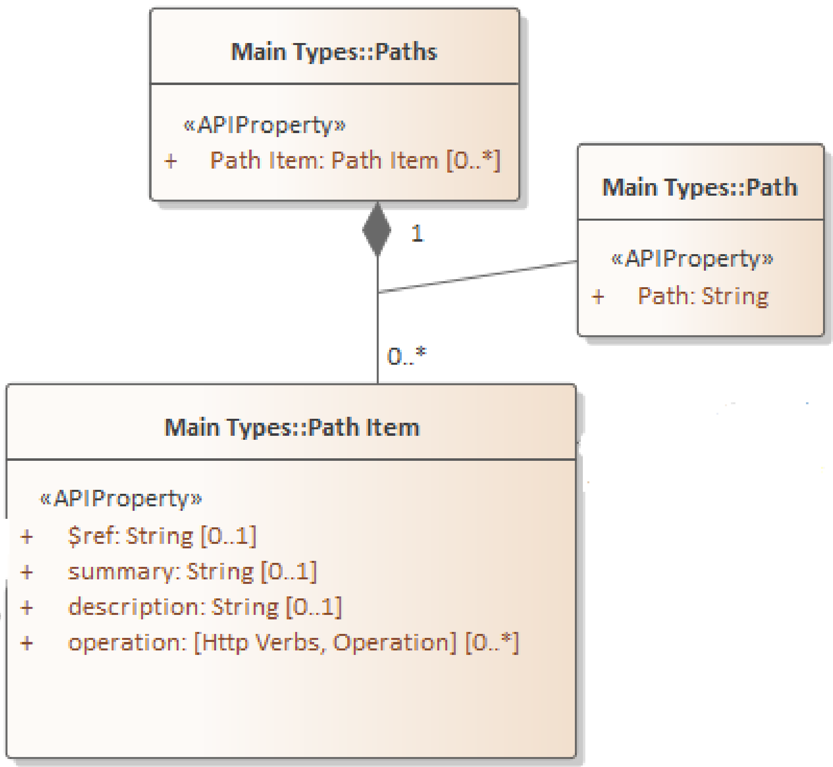


Figure 5. Use of Path association

The Paths Object in the OpenAPI specification contains a Field Pattern as an input variable name, therefore the specification is using a variable name as a typical variable with a Path Item Object as its type. This pattern does not map to the UML class diagram in the same way as the standard variable name and pattern. Association classes specialized from Path object (created for the model) allow for a path to be inserted into the model and document while maintaining the document hierarchy.

8.2.4.2. Association in Schema objects

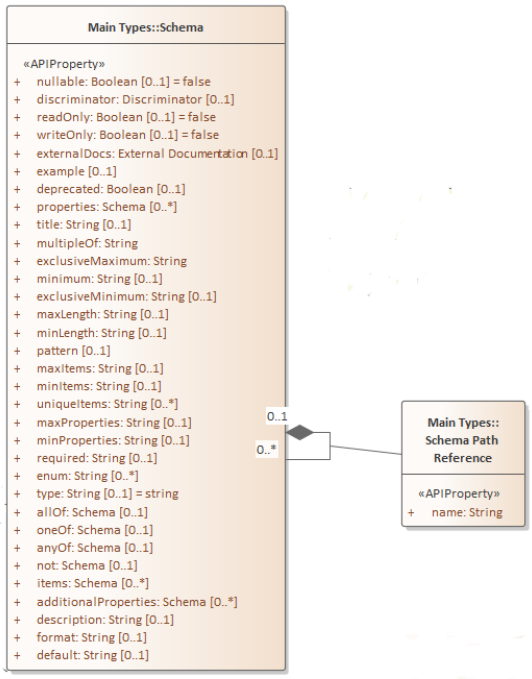

Main Types::Schema
«APIProperty»
+ nullable: Boolean [0..1] = false
+ discriminator: Discriminator [0..1]
+ readOnly: Boolean [0..1] = false
+ writeOnly: Boolean [0..1] = false
+ externalDocs: External Documentation [0..1]
+ example [0..1]
+ deprecated: Boolean [0..1]
+ properties: Schema [0..*]
+ title: String [0..1]
+ multipleOf: String
+ exclusiveMaximum: String
+ minimum: String [0..1]
+ exclusiveMinimum: String [0..1]
+ maxLength: String [0..1]
+ minLength: String [0..1]
+ pattern [0..1]
+ maxItems: String [0..1]
+ minItems: String [0..1]
+ uniqueItems: String [0..*]
+ maxProperties: String [0..1]
+ minProperties: String [0..1]
+ required: String [0..1]
+ enum: String [0..*]
+ type: String [0..1] = string
+ allOf: Schema [0..1]
+ oneOf: Schema [0..1]
+ anyOf: Schema [0..1]
+ not: Schema [0..1]
+ items: Schema [0..*]
+ additionalProperties: Schema [0..*]
+ description: String [0..1]
+ format: String [0..1]
+ default: String [0..1]
0..1
0..*
Main Types:: Schema Path Reference
«APIProperty»
+ name: String

Figure 6. Use of Schema self-referencing association

The Schema object is self-referential in multiple variables, for example, the properties variable is also an array of Schema objects. In other words, Schema Objects can have properties that are Schema Objects that can also have properties that are also schema objects. If a Schema Object is referenced from the Schemas aspect of the Components Object, then there is a typical name, value attribute entry to hold the schema name and the specific Schema Object class. However, when the schema is self-referencing, there is no place (in UML) to hold this information. Therefore, an Association Class is used to rectify this shortcoming. This is particularly salient in the Properties and Items variables. Another complication to this is that all Schema Objects can also hold References and any mixture of the two.

Schemas in OpenAPI rely on JSON schema as a base standard with a list of extensions relayed in the specification. Therefore, the UML representation of the schema object contains the JSON schema variables but with the OpenAPI constraints included.

8.2.4.3. Association in Callbacks objects

The association class is in callback objects to account for the attribute name linking callback to Path Item object being a variable in the specification named {expression}. An association class is used to intercept the relationship and insert a String type conforming to an expression. This item in the metamodel is not used in the implementation as it is not used explicitly in OGC API - Features, but remains for reference.

8.3. OGC API - Features - Part 1: Core

The approach to implementing the OGC API - Features standard is to utilize the metamodel via specialization of classes. This strategy provides ShapeChange with a generalized class to apply business rules to. For example, ShapeChange knows that the LandingPage class is a specialization of OpenAPI Object and therefore should be placed as the entry and top-level class for the model. Likewise, a class that inherits from a Map type will have a name/value pair. The name should be inserted at the appropriate place in the target document with the aspects of the type mapped to value placed as nested attributes. As the map type is specialized, the business rules can also be used control the vocabulary of the value type.

The OGC API - Features - Part 1: Core model follows the metamodel as closely as possible and all classes included are specializations of the metamodel classes. The class names are irrelevant to the modeling process and ShapeChange does not use them. However, the classes are named according to a convention that provides the class type initially and then naming according to where the class sits in the hierarchy, this is done for traceability purposes.

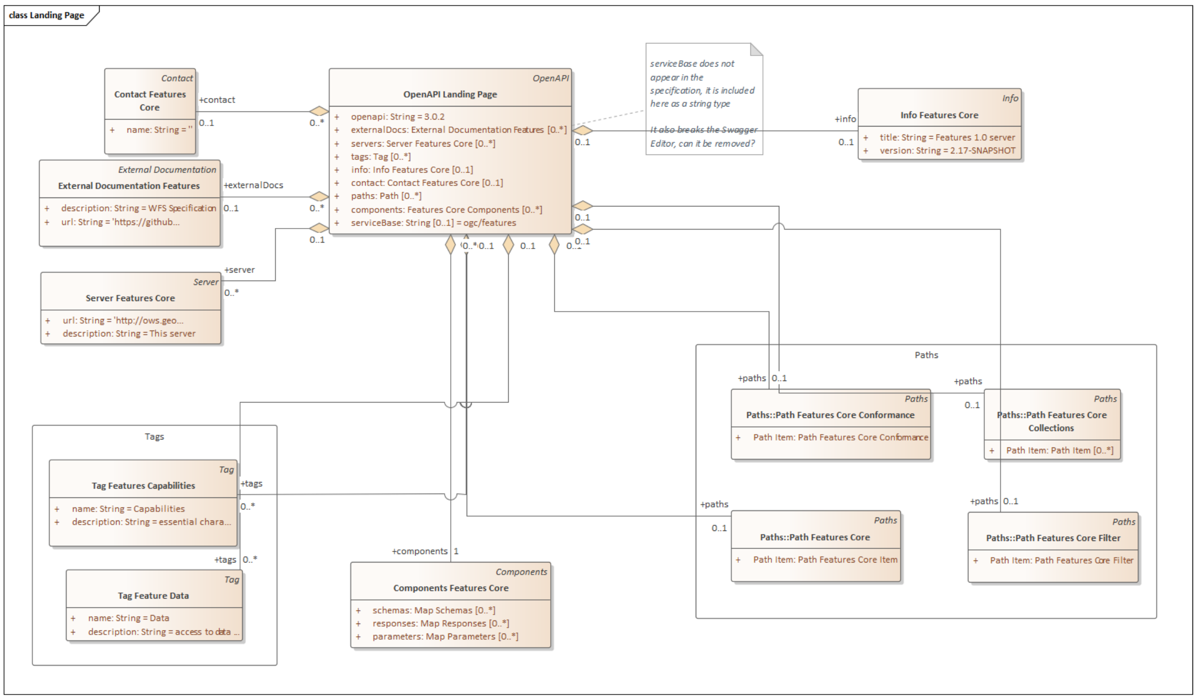
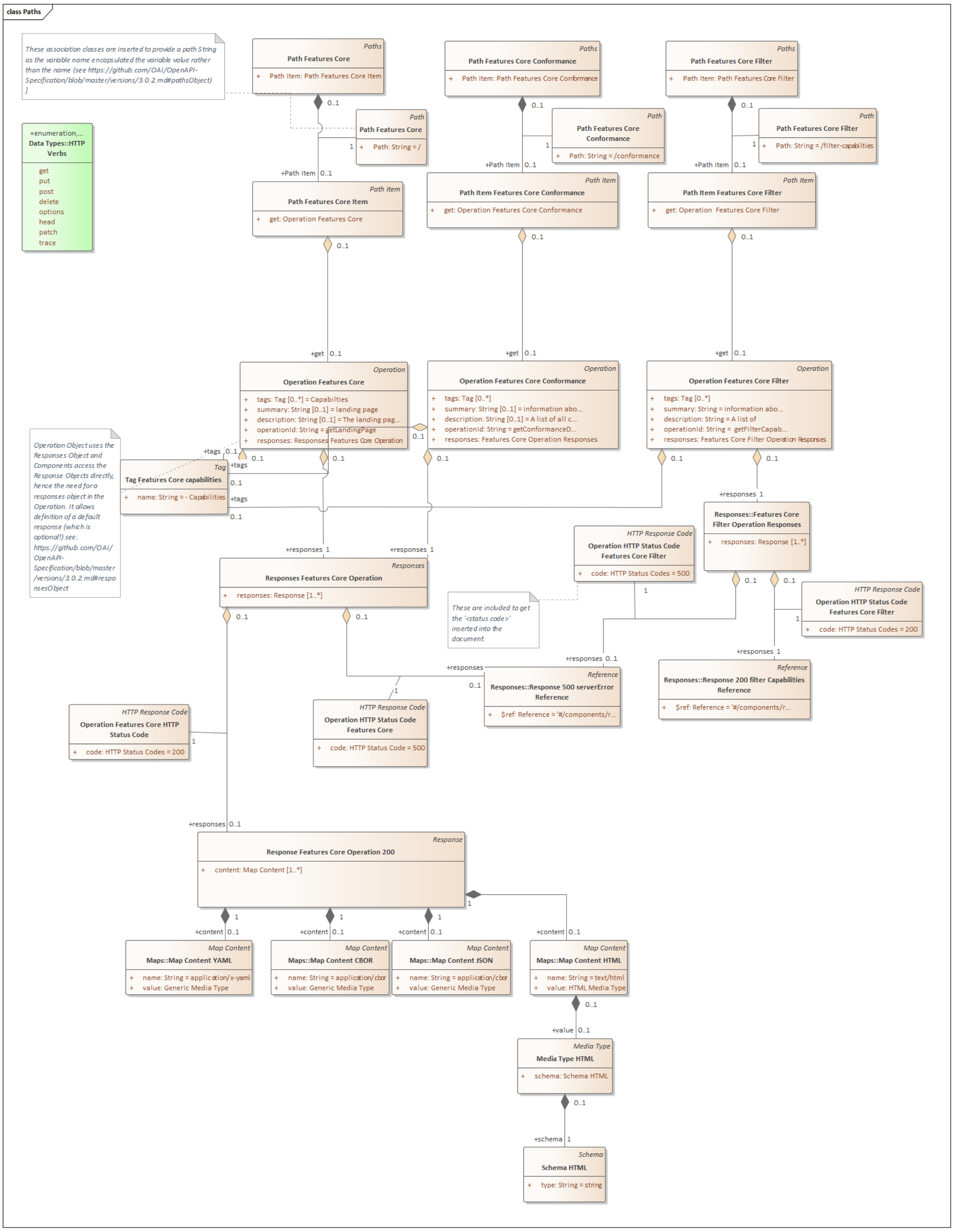

class Landing Page
Contact
Contact Features Core
+ name: String = ''
+contact
0..1
0..*
OpenAPI
OpenAPI Landing Page
+ openapi: String = 3.0.2
+ externalDocs: External Documentation Features [0..*]
+ servers: Server Features Core [0..*]
+ tags: Tag [0..*]
+ info: Info Features Core [0..1]
+ contact: Contact Features Core [0..1]
+ paths: Path [0..*]
+ components: Features Core Components [0..*]
+ serviceBase: String [0..1] = ogc/features
serviceBase does not appear in the specification, it is included here as a string type
~~It also breaks the Swagger~~ Editor, can it be removed?
Info
Info Features Core
+ title: String = Features 1.0 server
+ version: String = 2.17-SNAPSHOT
+info
External Documentation
External Documentation Features
+ description: String = WFS Specification
+ url: String = 'https://github...
+externalDocs
Server
Server Features Core
+ url: String = 'http://ows.geo...
+ description: String = This server
+server
Tags
Tag
Tag Features Capabilities
+ name: String = Capabilities
+ description: String = essential chara...
+tags
Tag
Tag Feature Data
+ name: String = Data
+ description: String = access to data ...
Components
Components Features Core
+ schemas: Map Schemas [0..*]
+ responses: Map Responses [0..*]
+ parameters: Map Parameters [0..*]
+components
1
Paths
Paths::Path Features Core Conformance
+ Path Item: Path Features Core Conformance
Paths::Path Features Core Collections
+ Path Item: Path Item [0..*]
Paths::Path Features Core
+ Path Item: Path Features Core Item
Paths::Path Features Core Filter
+ Path Item: Path Features Core Filter
+paths

Figure 7. Landing Page

The entire model is very large containing many classes and was built using EA.

8.3.1. Paths

Paths have been split into two diagrams for readability. The collections-based paths are separated into a different diagram.


class Paths
«enumeration,... Data Types::HTTP Verbs
get
put
post
delete
options
head
patch
trace
Paths
Path Features Core
+ Path Item: Path Features Core Item
Path
Path Features Core
+ Path: String = /
Path Item
Path Features Core Item
+ get: Operation Features Core
Paths
Path Features Core Conformance
+ Path Item: Path Features Core Conformance
Path
Path Features Core Conformance
+ Path: String = /conformance
Path Item
Path Item Features Core Conformance
+ get: Operation Features Core Conformance
Paths
Path Features Core Filter
+ Path Item: Path Features Core Filter
Path
Path Features Core Filter
+ Path: String = /filter-capabilties
Path Item
Path Item Features Core Filter
+ get: Operation Features Core Filter
Operation
Operation Features Core
+ tags: Tag [0..*] = Capabilties
+ summary: String [0..1] = landing page
+ description: String [0..1] = The landing pag...
+ operationId: String = getLandingPage
+ responses: Responses Features Core Operation
Operation
Operation Features Core Conformance
+ tags: Tag [0..*]
+ summary: String [0..1] = information abo...
+ description: String [0..1] = A list of all c...
+ operationId: String = getConformanceD...
+ responses: Features Core Operation Responses
Operation
Operation Features Core Filter
+ tags: Tag [0..*]
+ summary: String = information abo...
+ description: String = A list of
+ operationId: String = getFilterCapab...
+ responses: Features Core Filter Operation Responses
Tag
Tag Features Core capabilities
+ name: String = - Capabilities
Responses
Responses Features Core Operation
+ responses: Response [1..*]
HTTP Response Code
Operation HTTP Status Code Features Core Filter
+ code: HTTP Status Codes = 500
Responses::Features Core Filter Operation Responses
+ responses: Response [1..*]
HTTP Response Code
Operation HTTP Status Code Features Core Filter
+ code: HTTP Status Codes = 200
Reference
Responses::Response 500 serverError Reference
+ $ref: Reference = '#/components/r...
Reference
Responses::Response 200 filter Capabilities Reference
+ $ref: Reference = '#/components/r...
HTTP Response Code
Operation Features Core HTTP Status Code
+ code: HTTP Status Codes = 200
HTTP Response Code
Operation HTTP Status Code Features Core
+ code: HTTP Status Code = 500
Response
Response Features Core Operation 200
+ content: Map Content [1..*]
Map Content
Maps::Map Content YAML
+ name: String = application/x-yaml
+ value: Generic Media Type
Map Content
Maps::Map Content CBOR
+ name: String = application/cbor
+ value: Generic Media Type
Map Content
Maps::Map Content JSON
+ name: String = application/cbor
+ value: Generic Media Type
Map Content
Maps::Map Content HTML
+ name: String = text/html
+ value: HTML Media Type
Media Type
Media Type HTML
+ schema: Schema HTML
Schema
Schema HTML
+ type: String = string
+Path Item 0..1
+get 0..1
+tags 0..1
+responses 1
+responses 0..1
+content 0..1
+value 0..1
+schema 1

These association classes are inserted to provide a path String as the variable name encapsulated the variable value rather than the name (see https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#pathsObject)]

Operation Object uses the Responses Object and Components access the Response Objects directly, hence the need for a responses object in the Operation. It allows definition of a default response (which is optional!) see: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/3.0.2.md#responsesObject

These are included to get the '<status code>' inserted into the document.

Figure 8. Paths

8.3.1.1. Collections

Collections form the bulk of the paths, therefore they have been included separately.

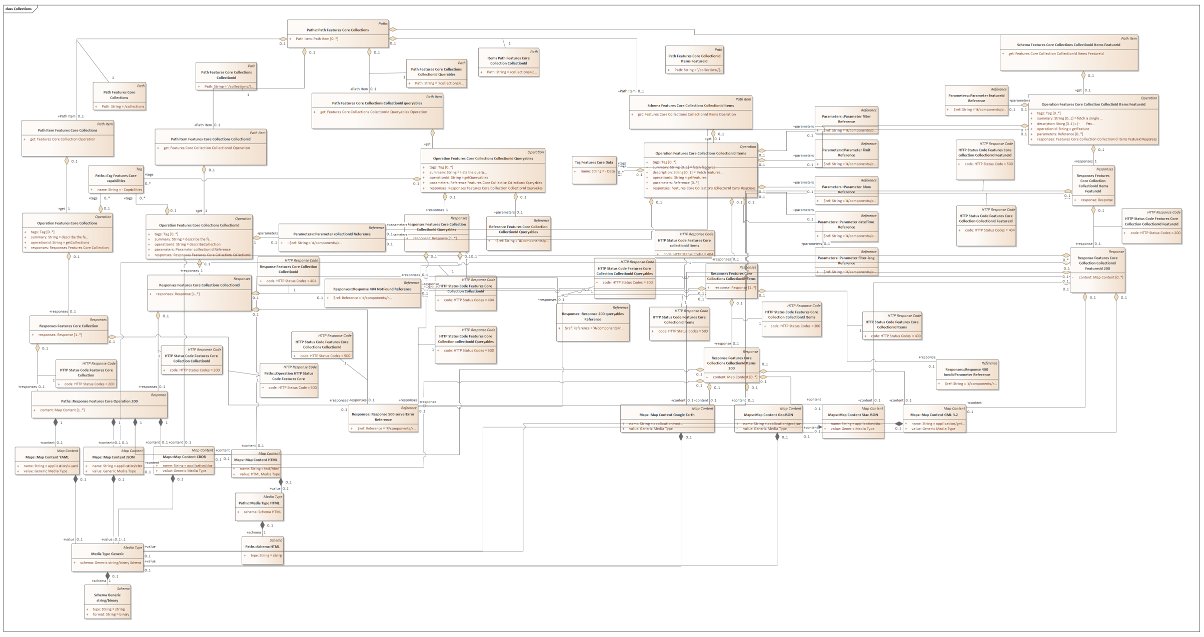


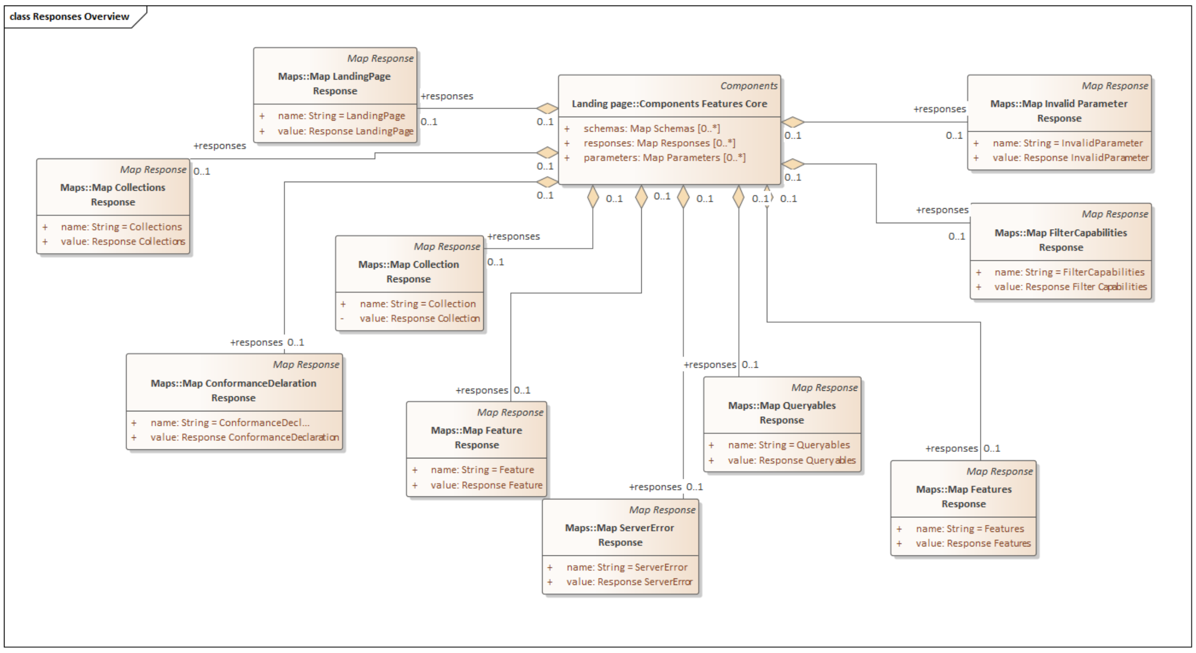
Figure 9. Collections

8.3.2. Components

The components aspect is a major part of the OpenAPI hierarchy as components contain all of structures and definitions required to build API queries from building blocks as well as the response type the client should expect. Many of the response types and parameters can now be defined as references, thus removing the complex structure from the model.

8.3.2.1. Responses

The responses are mapped out and sit in the *Components* aspect of the OpenAPI document. The responses, schemas and parameters are linked from their component definitions to their instantiations via a reference object. The Figure 10 provides the top level responses and their relationships to their *Map* object.



class Responses Overview
Components
Landing page::Components Features Core
+ schemas: Map Schemas [0..*]
+ responses: Map Responses [0..*]
+ parameters: Map Parameters [0..*]
Map Response
Maps::Map LandingPage Response
+ name: String = LandingPage
+ value: Response LandingPage
Maps::Map Collections Response
+ name: String = Collections
+ value: Response Collections
Maps::Map Collection Response
+ name: String = Collection
- value: Response Collection
Maps::Map ConformanceDelaration Response
+ name: String = ConformanceDecl...
+ value: Response ConformanceDeclaration
Maps::Map Feature Response
+ name: String = Feature
+ value: Response Feature
Maps::Map ServerError Response
+ name: String = ServerError
+ value: Response ServerError
Maps::Map Queryables Response
+ name: String = Queryables
+ value: Response Queryables
Maps::Map Features Response
+ name: String = Features
+ value: Response Features
Maps::Map Invalid Parameter Response
+ name: String = InvalidParameter
+ value: Response InvalidParameter
Maps::Map FilterCapabilities Response
+ name: String = FilterCapabilities
+ value: Response Filter Capabilities
+responses
0..1

Figure 10. Responses Overview

Figure 11 provides the detail for each of the responses. The responses are instantiated in the *Operation* aspect of the *Paths* class.

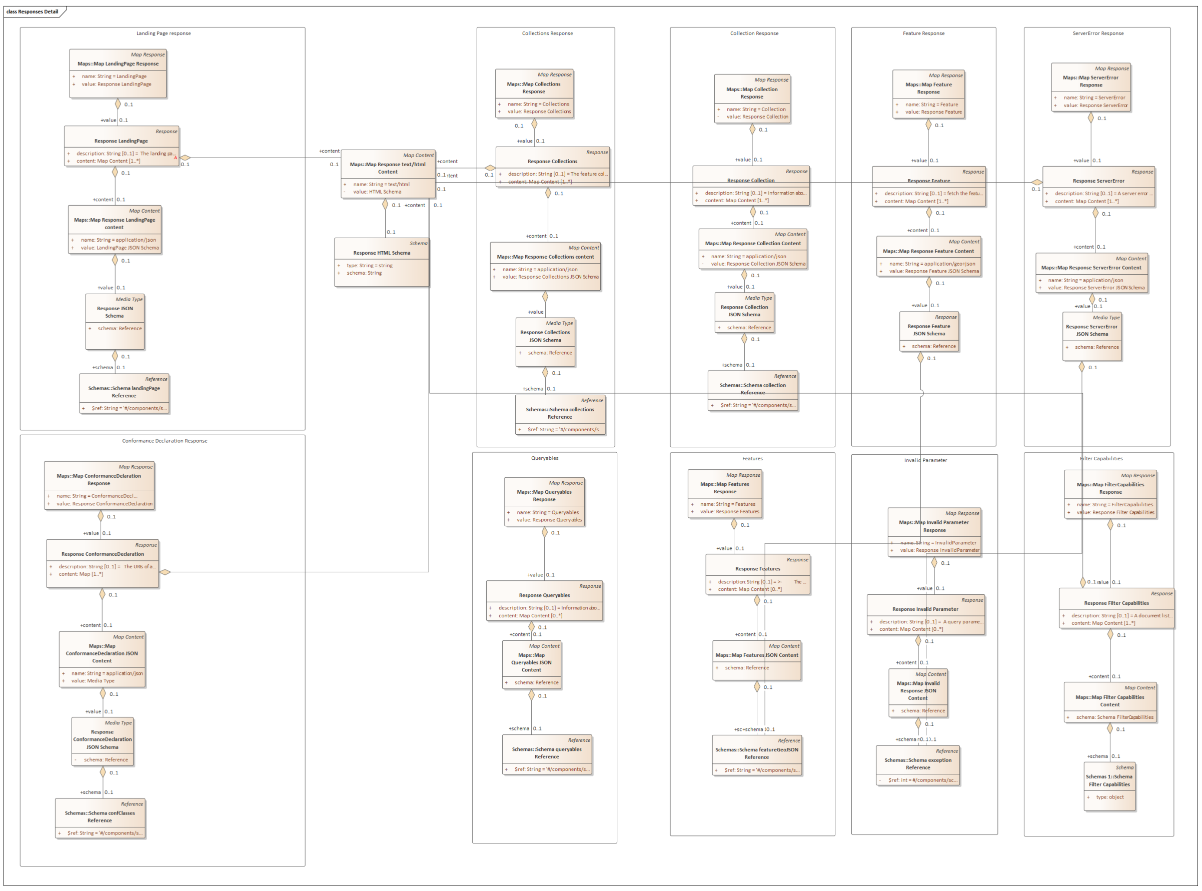


Figure 11. Responses Detail

As mentioned previously, the Map object is specialized to a *Response Map* object to control for the value type. Many of the response returns rely on references that are re-used throughout the Responses classes.

8.3.2.2. Parameters

Parameters are components that often contain schemas, again they are tied to the Paths aspect via a Reference. The schemas are self-referential and contain schemas within schemas. This was the most difficult aspect of the standard to model due to multiple nested schemas.

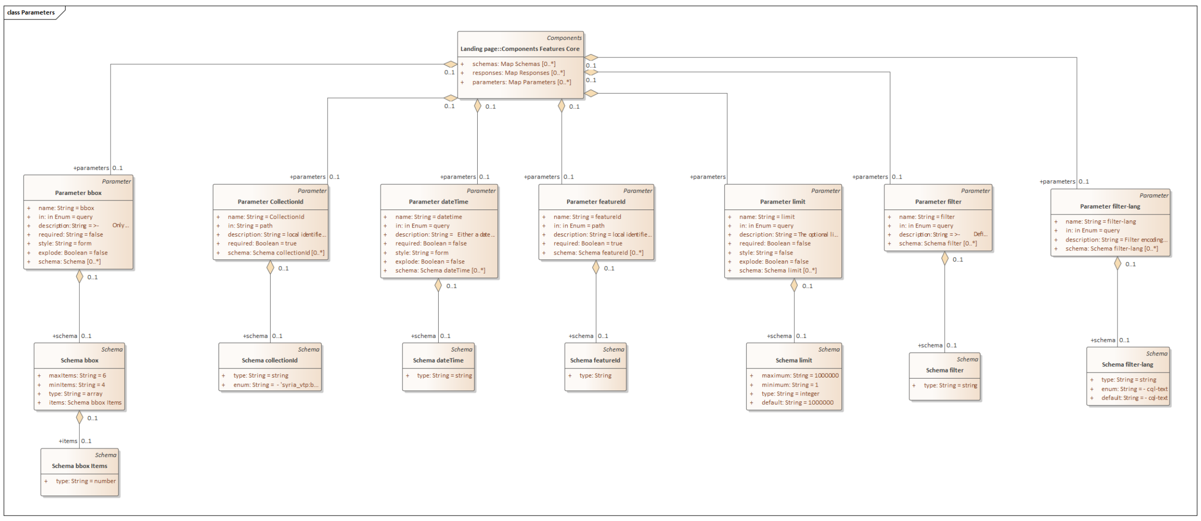


Figure 12. Parameters

The *Parameter* objects are far simpler than the Response objects as they simply define the inputs for each parameter. A geospatially relevant parameter is the BBOX, where the maximum and minimum number of entries that are valid are 4 and 6 respectively to reflect the standardized manner that BBOXs are defined.

8.3.2.3. Schemas

Schemas are by far the most complex and numerous of the components and are split into 9 diagrams, there is one included for reference here.

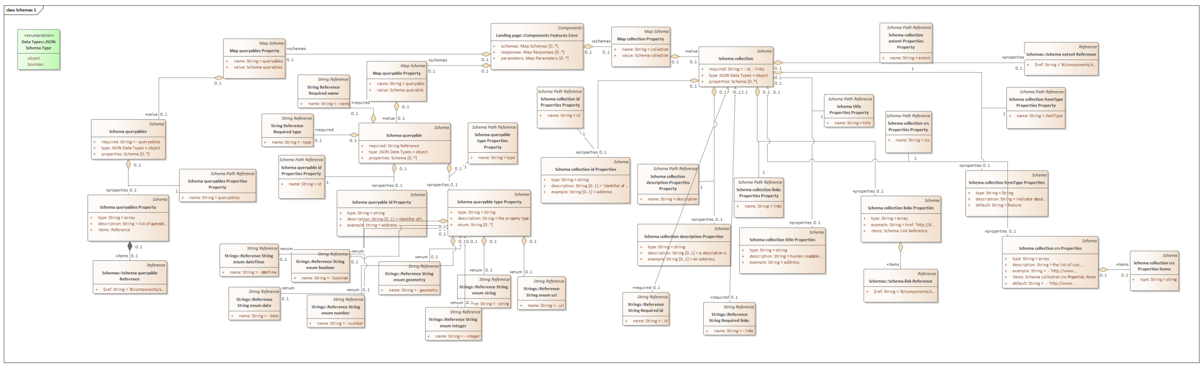


Figure 13. Schemas example

Schemas become more complex as the parameters that are included also contain attributes of type schemas. This potentially means that attributes within the main schema types contain schemas with further schemas nested further down the tree. Although conceptually easy in JSON, this does make for a sizable UML model when fully modeled.

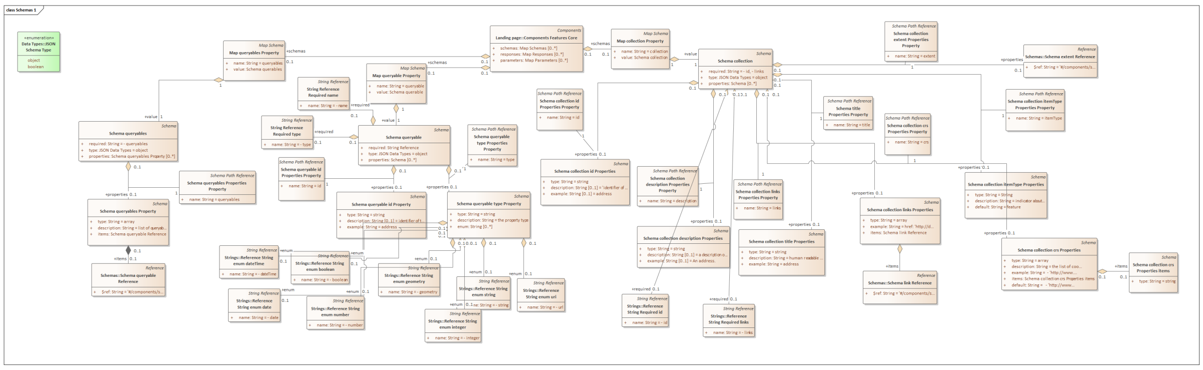


Figure 14. Schemas example

The schema type is a prime example of use of an association class to intercept relationships. At the top level of schema, when instantiated via the Schemas object, the relationship calls for a Schema Map object, which allows for a name, value pairing. However, when a Schema Object is instantiated within a higher Schema Object through, for example the Properties Object, another Schema object is called for without the use of the Schema Map type. Therefore, there is no place to model the attribute name (properties), the schema name (queryables, for example) and the schema class being referred to (Schema queryables property). Therefore, an association class is inserted into the relationship between the attribute properties and the class Schema *queryables property*.

8.3.3. References

As mentioned previously, references tie definitions of components to instantiations.

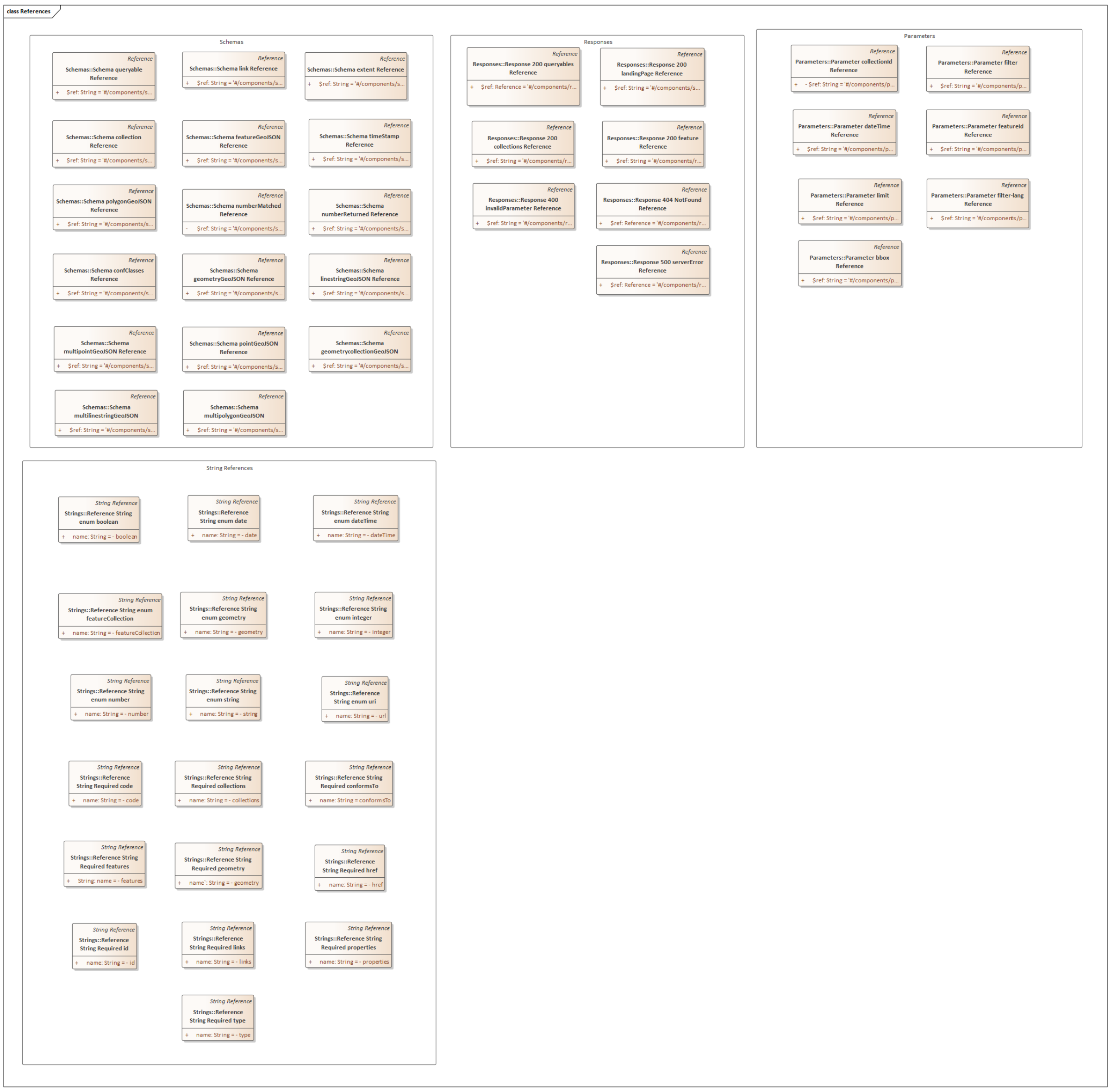


Figure 15. References

The reference type was included to utilize entries across the entire UML model. For example, the reference to *conformance classes* provides the modeler with the ability to hold the references in a separate pool for implementation across aspects of the model and across other standards should they be modeled in the future. A stripped-down version of OGC API - Features could contain references instead of formal definitions of most schemas and parameters.

Chapter 9. ShapeChange Implementation

ShapeChange is a software application that converts UML models to different target outputs. These targets could be XML, JSON or a different format with validation and encoding rules applied at the point of conversion. ShapeChange already has many targets and at the time Testbed 16 was completed also had a preliminary implementation for JSON schema on a separate branch. However, ShapeChange does not have a target configured for OpenAPI interfaces. Therefore, the primary development work documented here was concerned with developing the software to address the OpenAPI target.

The purpose of using ShapeChange in the OpenAPI Thread of Testbed-16 was to convert a UML representation of an OpenAPI definition into the JSON encoding. In a real-world environment, the user could then take the generated JSON document and stub the required classes for implementation. The test dataset for the implementation was the OGC API - Features - Part 1: Core (described in previous sections). The changes to ShapeChange are also not designed to solve for the generic standard case as the development work in the Testbed was largely a research exercise to understand the process of using the MDA approach for defining interfaces.

ShapeChange connects to an EA file and creates a ShapeChange internal representation of the model in SCXML and uses this as a baseline to access the variables from the UML model. The structure of the SCXML enables all relevant relationships, classes, attributes and variables to be captured for transformation into the target apart from where mentioned. The development delta is manipulation of SCXML to produce the correct target. Modifying ShapeChange’s core code and functionality is out of scope.

9.1. Initial Configuration

The ShapeChange configuration can be complex, to keep the configuration simple, the following variables were set:

- The SCXML input file name.
- The *home* package, i.e. the entry point into the model.
- The output Path for the OpenAPI JSON file.

In a real world scenario, one could use more of the configuration variables to control for vocabularies and other, extended aspects of JSON schema.

Below is the configuration file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ShapeChangeConfiguration xsi:schemaLocation="http://www.interactive-
instruments.de/ShapeChange/Configuration/1.1
config/schema/ShapeChangeConfiguration.xsd" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:sc="http://www.interactive-
instruments.de/ShapeChange/Configuration/1.1" xmlns="http://www.interactive-
instruments.de/ShapeChange/Configuration/1.1" xmlns:xi=
"http://www.w3.org/2001/XInclude">
  <input id="model">
    <parameter name="inputModelType"  value="SCXML"/>
    <parameter name="inputFile" value=".\results\modelexport\INPUT\schema_export.xml
"/>
    <parameter value="true" name="publicOnly"/>
    <parameter value="disabled" name="checkingConstraints"/>
    <parameter value="true" name="sortedSchemaOutput"/>
    <xi:include href="http://shapechange.net/resources/config/StandardAliases.xml"/>
    <packages>
      <PackageInfo packageName="OGC API - Features - Part 1: Core" nsabr="app"
xsdDocument="http://shapechange.net/resources/schema/ShapeChangeConfiguration.xsd"
version="0.01"/>
    </packages>
  </input>
  <log>
    <parameter value="INFO" name="reportLevel"/>
    <parameter value="test/SCXML/log.xml" name="logFile"/>
  </log>
  <targets>
    <Target class=
"de.interactive_instruments.ShapeChange.Target.HelyxOpenApi.OpenAPIEntry" mode=
"enabled" inputs="model">
     <targetParameter name="outputDirectory" value="results/openapi"/>
     <targetParameter name="outputFilename" value="openapi_instantiable.json"/>
    </Target>
  </targets>
</ShapeChangeConfiguration>
```

9.2. Target Generation Rules

The plugin to ShapeChange enables generation of OpenAPI 3.0 targets from the UML Class diagram described in this ER. However, generation of targets using UML required design decisions made in the modeling phase. One of the heuristics applied to the SCXML representation of the UML model was that attributes and values were treated as key:value pairs. This heuristic solves many of the issues with the ShapeChange implementation, with exceptions created on a meta-class basis.

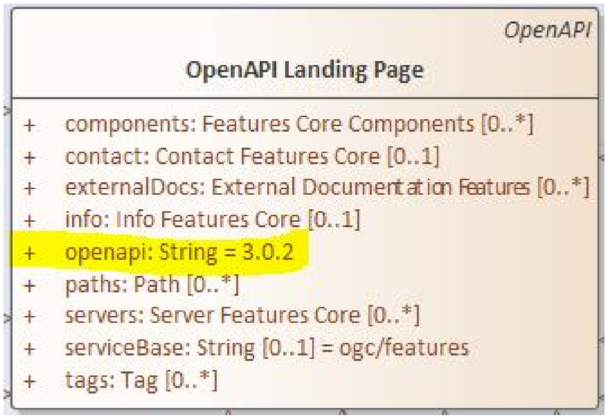
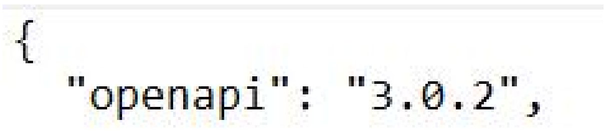

OpenAPI
OpenAPI Landing Page
+ components: Features Core Components [0..*]
+ contact: Contact Features Core [0..1]
+ externalDocs: External Documentation Features [0..*]
+ info: Info Features Core [0..1]
+ openapi: String = 3.0.2
+ paths: Path [0..*]
+ servers: Server Features Core [0..*]
+ serviceBase: String [0..1] = ogc/features
+ tags: Tag [0..*]

Figure 16. UML representation of Landing Page class



```
{
   "openapi": "3.0.2",
```

Figure 17. JSON representation of the OpenAPI attribute

As is often the case that the attribute value is another class the class name makes up the Key aspect of the relationship and the class contents is the value. As mentioned previously, this is not the suggested solution for an operational version of this software. Business rules should be based upon the meta-class. This approach solves many of the cases, with a notable exception of the schemas, which has many recursive relationships. In this instance, the name is ignored and the relationship is defined through the association class attribute value. In a second iteration of the implementation, the class name would not be used and the generation rules would be derived using a rules engine linked to the generalized classes and thus the specification

The current version of the model uses the convention of name and value as a key/value pair. Where this is the case, the JSON has the key aspect equal to the name and the contents of the value field is made of the *value* attribute in the UML.

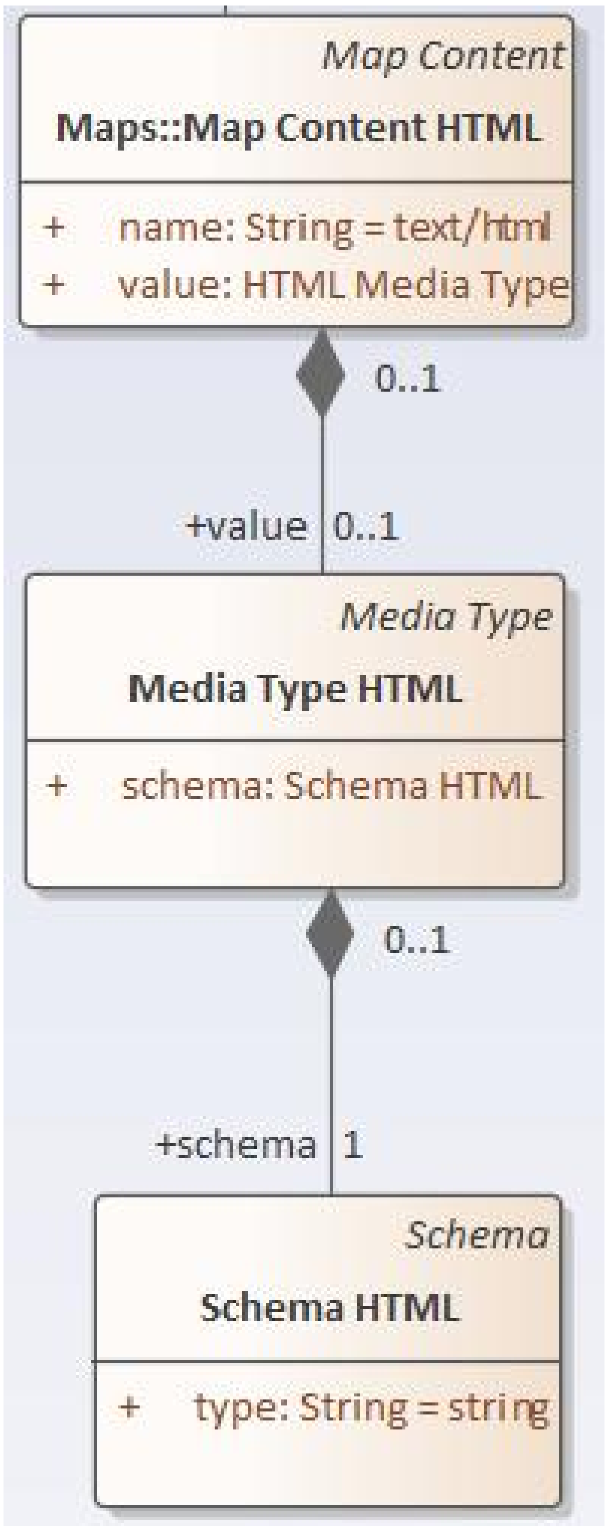
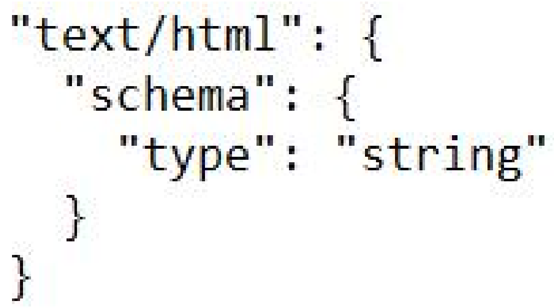


Figure 18. UML representation of a name/value pair



```
"text/html": {
  "schema": {
    "type": "string"
  }
}
```

Figure 19. JSON representation of a name/value pair as a key/value pair

However, if there is an attribute with a *name* field, then it is treated as either a simple attribute, or a class as described in the previous paragraphs.

Many of the classes contain an aggregation property role for several classes related to a single class at a higher level. When this is the case, the contents of the associated classes all appear at the same level in the output JSON.

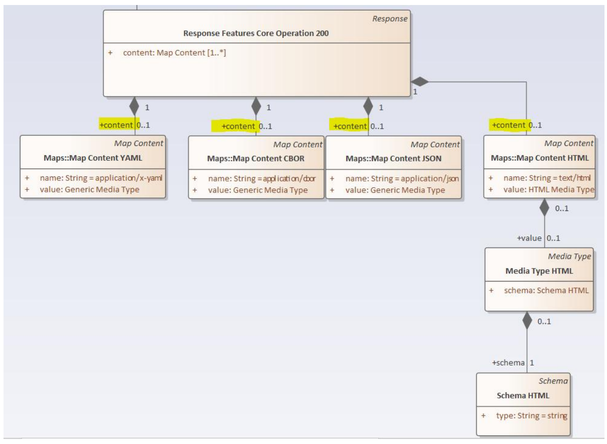
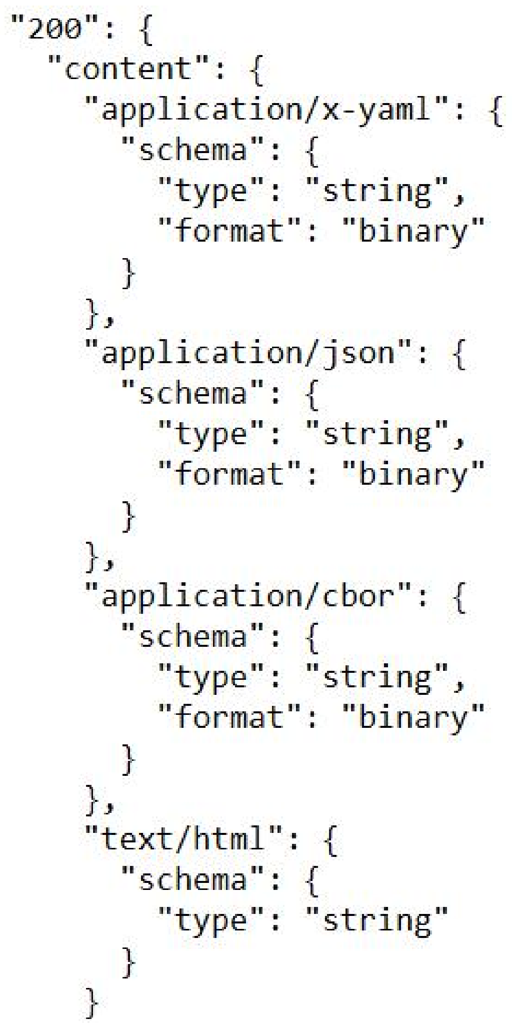


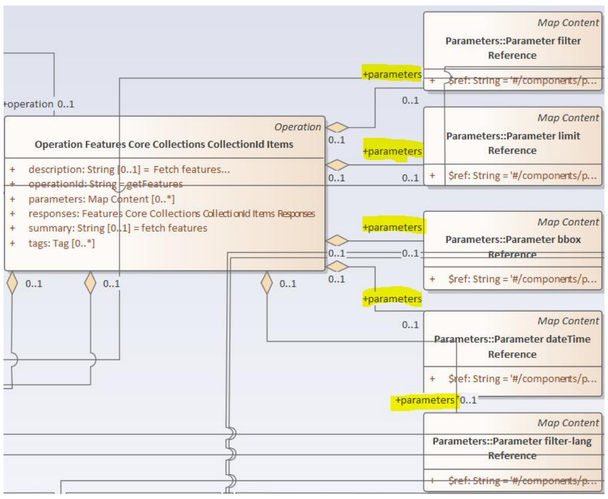
Figure 20. UML representation of multiple classes aggregated to a single class



```
"200": {
  "content": {
    "application/x-yaml": {
      "schema": {
        "type": "string",
        "format": "binary"
      }
    },
    "application/json": {
      "schema": {
        "type": "string",
        "format": "binary"
      }
    },
    "application/cbor": {
      "schema": {
        "type": "string",
        "format": "binary"
      }
    },
    "text/html": {
      "schema": {
        "type": "string"
      }
    }
```

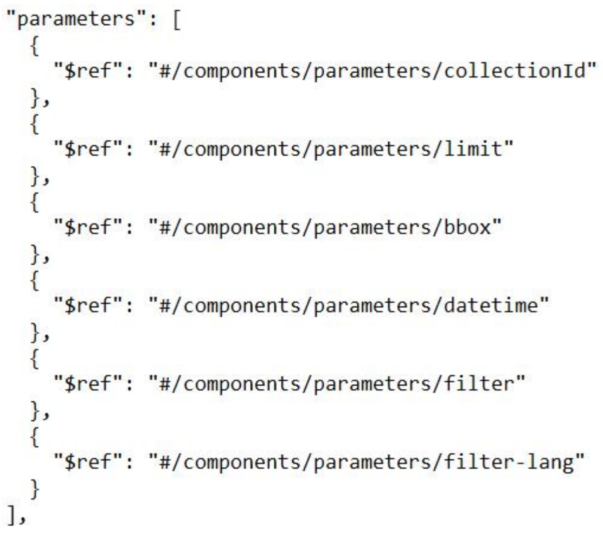
Figure 21. JSON representation of multiple classes aggregated to a single class

The described approach works as a general rule except in certain circumstances, for example if the association property role is equal to *parameters* then the output JSON content will be inside a JSON array if there is more than class with the same association property role name to a higher level class.



Map Content
Parameters::Parameter filter Reference
+ $ref: String = '#/components/p...
+parameters
0..1
+operation 0..1
Operation
Operation Features Core Collections CollectionId Items
+ description: String [0..1] = Fetch features...
+ operationId: String = getFeatures
+ parameters: Map Content [0..*]
+ responses: Features Core Collections CollectionId Items Responses
+ summary: String [0..1] = fetch features
+ tags: Tag [0..*]
Parameters::Parameter limit Reference
Parameters::Parameter bbox Reference
Parameters::Parameter dateTime Reference
Parameters::Parameter filter-lang Reference

Figure 22. UML representation the parameters class



```
"parameters": [
  {
    "$ref": "#/components/parameters/collectionId"
  },
  {
    "$ref": "#/components/parameters/limit"
  },
  {
    "$ref": "#/components/parameters/bbox"
  },
  {
    "$ref": "#/components/parameters/datetime"
  },
  {
    "$ref": "#/components/parameters/filter"
  },
  {
    "$ref": "#/components/parameters/filter-lang"
  }
],
```

Figure 23. JSON representation of the parameters class

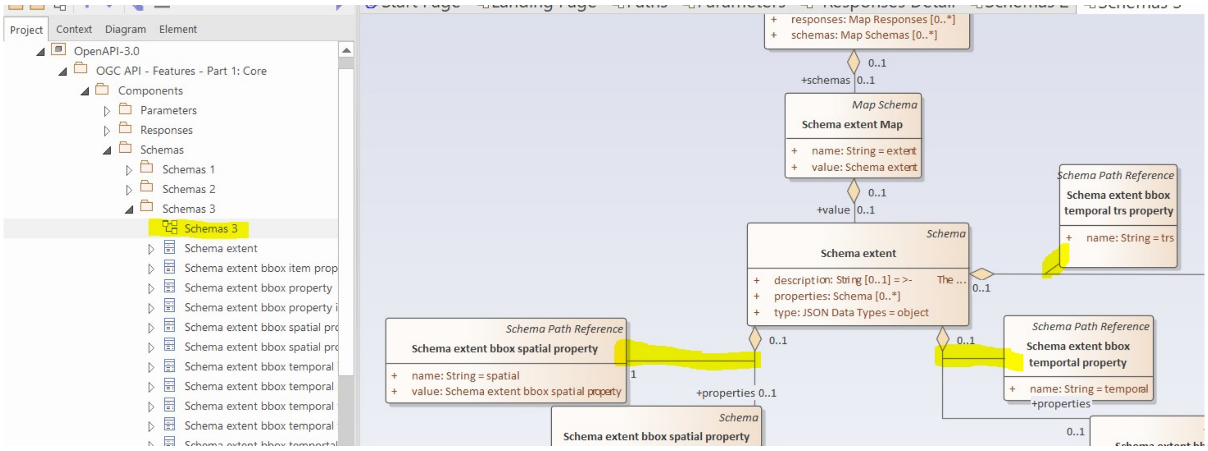
Enumerations are dealt with by creation of a JSON array to contain the values and the responses and currently the response codes are extracted from the text of a class joining a response.

There are several other general rules that are implemented as part of the ShapeChange conversion process. For example, the ShapeChange configuration defines four classes to search through in sequence; LandingPage, Paths, Components and Schemas. This approach generates an output that resembles an OpenAPI document with minor errors.

As described in Testbed-14 JSON schemas ER, inheritance is managed within the schema aspect using the JSON Schema type *allof* (https://json-schema.org/understanding-json-schema/reference/combining.html#allof). This mechanism for managing this is well defined in the JSON schema, Testbed-14 and the recently concluded UGAS Pilot.

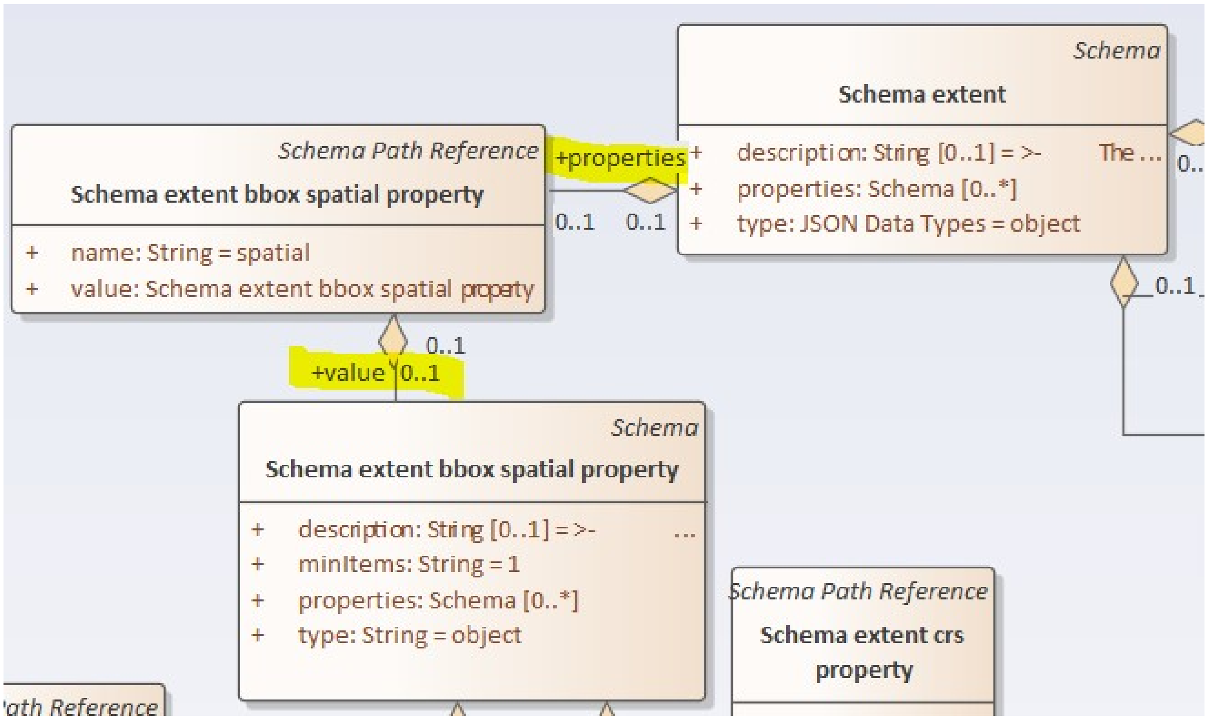
9.3. Association Classes

Association classes are used in the UML model to inject text into the JSON document. Examples include schemas where the term schema is not used, paths where the attribute name is also a variable, and responses where the attribute name is a controlled vocabulary. ShapeChange recognizes the association classes, but it does not recognize the relationship that the association class is used to intercept. Therefore, approaches such as looking for the response code in a class name, mentioned in the previous section, were adopted to create the OpenAPI JSON output. Another approach to this is to use a name/value pair relationship via a qualifier, or to split the association class into a normal class with two separate relationships.



Project
Context
Diagram
Element
OpenAPI-3.0
OGC API - Features - Part 1: Core
Components
Parameters
Responses
Schemas
Schemas 1
Schemas 2
Schemas 3
Schemas 3
Schema extent
Schema extent bbox item prop
Schema extent bbox property
Schema extent bbox property i
Schema extent bbox spatial pro
Schema extent bbox spatial pro
Schema extent bbox temporal
Schema extent bbox temporal
Schema extent bbox temporal
Schema extent bbox temporal
+ responses: Map Responses [0..*]
+ schemas: Map Schemas [0..*]
0..1
+schemas 0..1
Map Schema
Schema extent Map
+ name: String = extent
+ value: Schema extent
0..1
+value 0..1
Schema Path Reference
Schema extent bbox temporal trs property
+ name: String = trs
Schema
Schema extent
+ description: String [0..1] = >- The ...
+ properties: Schema [0..*]
+ type: JSON Data Types = object
0..1
Schema Path Reference
Schema extent bbox spatial property
+ name: String = spatial
+ value: Schema extent bbox spatial property
1
0..1
0..1
Schema Path Reference
Schema extent bbox temporal property
+ name: String = temporal
+properties
+properties 0..1
0..1
Schema
Schema extent bbox spatial property

Figure 24. UML representation of relationship using an association class



Schema
Schema extent
+ description: String [0..1] = >- The ...
+ properties: Schema [0..*]
+ type: JSON Data Types = object
0..
0..1
Schema Path Reference
Schema extent bbox spatial property
+ name: String = spatial
+ value: Schema extent bbox spatial property
+properties
0..1
0..1
0..1
+value 0..1
Schema
Schema extent bbox spatial property
+ description: String [0..1] = >- ...
+ minItems: String = 1
+ properties: Schema [0..*]
+ type: String = object
Schema Path Reference
Schema extent crs property
ath Reference

Figure 25. UML representation of a relationship using a class with separate connectors

9.4. Results

The results of the MDA process including the UML models and implementations are not perfect. Much of the generated OpenAPI document is conformant and the base structure is correct. However, there are issues with the more complex aspects of the model including schemas. Additionally, the system was only tested using an implementation of OGC API - Features – Part 1: Core. There are bound to be more issues with the output of the MDA process, should the entire specification be under scrutiny. The output XML document can be found in the appendix OGC API Features Part1:Core output (JSON).

Chapter 10. Discussion

Using the MDA approach to generate OpenAPI documents, specifically for OGC API Standards, was successful. The tested process produced a JSON-encoded OpenAPI definition document from a UML model. However, there are lessons to be learned from the exercise, and the focus of future work should shift from ability to utility.

OGC Standards and OpenAPI documents can be represented using a UML model and the approach described in this ER. Design decisions were made and documented throughout the exercise. However, questions remain as to whether representing an interface as a UML model is a desirable course of action. One aspect of the process that is immediately apparent is the complexity, size and time investment required to produce UML models for these interfaces. Using the MDA approach seems to enforce rigor at the detriment of speed. This is certainly true when creating a model from scratch.

Another consideration is a wider OGC concern: OGC is a standards body that produces open standards that are designed to be implemented as simply as possible with a view to remaining stable and interoperable over time. The current suite of approved and emerging OGC API Standards are documented using OpenAPI. This may not be true in perpetuity. There may come a time when standards development and definition will migrate to another chosen framework, specification or encoding. Therefore, standards could potentially be designed using a conceptual approach rather than an encoding based approach. A typical example of this is the relationship between the ISO 19115-1:2014 Geographic Metadata standard and ISO 19115-3:2016 the associated XML schema for the metadata standard. Both of the documents are standards. However an alternative encoding of ISO 19115 is feasible.

Perhaps the OGC should focus more on agreeing on a core conceptual model for a given standard. This could be done by starting with a conceptual model based on the requirements for a `feature service` and then look to derive implementable standards from that conceptual architecture. This is opposed to starting with a specification/encoding and translating older standards to have the same functionality. An additional observation is that the conceptual requirements for processing, disseminating and visualizing geospatial data over time remain the same within domains. However, the technology changes rapidly. MDA represents an approach to defining standards where the domain work remains reasonably static and the physical implementation of the conceptual model changes frequently.

The separation of conceptual aspects of a standard and their encoding is likely to become more salient as the OGC API suite of standards develop. An example of this is OGC API - Records, which is an emerging draft standard that is very similar to OGC API - Features though supporting access to metadata records. The conformance classes defined in the draft reflect those in the draft of OGC API – Common. However, it currently does not make a provision for typical metadata standards such as ISO 19115, nor does it define a JSON encoding for the standard, which would be in keeping with the rest of the OGC API suite.

10.1. Recommendations

The project largely verified the MDA approach to producing OpenAPI documents from UML. The recommendations from the work performed in Testbed-16 are:

- **The modeling process is very complex.** Reducing the complexity is necessary if the OGC wishes to pursue the MDA approach to generating APIs. Creation of the metamodel was very time consuming and may need to be reevaluated each time a new version of the OpenAPI specification is released. This also raises questions about versioning and different targets for different versions of the specification. A method for simplifying the modeling process is to create modeling artifacts that can be used in the definition of new standards or to update older standards when there is a version change. These building blocks should be created from OGC API - Common.
- **Consider other targets for generation.** For documenting OGC Web APIs, at this time OpenAPI is the specification/tool of choice. However, future interoperability could be improved through the MDA approach. This recommendation is an extension of thoughts expressed in the discussion section: De-coupling the conceptual aspects of a standard from an encoding while utilizing an automated process to generate artifacts directly from models is key to maintaining standards overtime. Once established the conceptual aspects of a standard do not change very frequently. It is the encoding or the presentation more widely of the conceptual aspects that changes frequently. Therefore, the MDA approach allows for maintenance of the conceptual requirements of the domain and generation into many artifacts. A typical example of this is OGC API - Processes, which at one point was a direct translation of Web Processing Service (WPS) version 2. The conceptual requirements of a processing interface have not changed, but the presentation has.
- **Understand the relationship between a standard and an encoding.** OpenAPI is a specification for machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services. A standard represents what is required in a domain. The OGC APIs should reflect this separation. Related to the previous recommendation, this suggests that the OGC should understand the relationship between a standard's domain and the standard's encoding.
- **The OpenAPI thread only considers going from OpenAPI to UML.** A converter should be written to generate UML from OpenAPI documents. This would help the OGC document all of their emerging standards in a recognized manner. The reverse translation from OGC API to UML would be useful for not only documentation purposes but also for verification and validation. Reverse engineering API documents to UML models would also allow for visualization of the API to determine the suitability of the interface to the domain and its compliance to OGC API rules.

10.2. Future Work

The aim of the Testbed-16 OpenAPI work was to understand the feasibility of the MDA approach to generating OpenAPI documents. The UML models generated using this process along with ShapeChange extension have largely validated this assumption. The UML models created in this project represent the API as faithfully as possible given that the specification is not based on UML. Likewise, the ShapeChange implementation takes the raw, unverified UML model and generates a JSON representation using the UML as is with some modifications in the ShapeChange code. However, further work is required to move the approach and demonstrator up the Technology Readiness Levels (TRL) where it could feasibly be used in the general case. Some ideas for future work are as follows:

- Investigate and implement a suitable way to record and apply the rules from the specification. The current approach makes some modifications to the input XML in order to achieve an output approaching a JSON representation of an OpenAPI document. However, future iterations should implement the rules of the specification for the generalized metamodel. A rules engine would potentially be able to perform these tasks.
- Add the OGC constraints to the process. The Testbed-16 work was interested in the OpenAPI specification. OGC standards have their own set of requirements and constraints beyond the OpenAPI specification that should be represented in the transformation aspect of the system.
- Investigate abstracting the approach to generating OpenAPI documents that include all JSON schemas. Only OpenAPI interface definitions have been considered in this project. However, the same approach can be applied to data models and other uses of JSON schemas.
- Create UML building blocks to represent common components of an OpenAPI interface. OGC API - Common is a good place to start with this work as it should contain the components common to OGC APIs. These building blocks would provide an interface designer with the required aspects of an OGC API and remove repetition and complexity from the design process.
- Create best practices for UML modeling to represent JSON schemas/OpenAPI interfaces. Although UML is a ratified standard with rules, the application of OpenAPI to UML can be done in several ways that are compliant with the UML standard. A best practices document would codify the recommended way to model OpenAPI documents and JSON more widely. This would simplify the implementation of the OpenAPI target transformation software as it would have to deal with fewer design methodologies.

Chapter 11. Conclusion

This project sought to use the MDA approach to generate OpenAPI interface definitions from UML class diagrams. The project was largely successful in that the process from start to finish was completed with much of the OpenAPI structure generated from a UML model. There were some shortcomings of the approach, notably some aspects of the UML document did not generate correctly as the objects within the specification required rules to validate and organize the information captured within the model to produce a compliant output. Overall, the project highlights the difference between domain knowledge, held within the UML model, and encoding, which is completed when the model is transformed into a particular target.

Appendix A: OGC API - Feature Part1:Core initial input document (YAML)

```
openapi: 3.0.2
info:
  title: Features 1.0 server
  contact:
    name: ''
  version: 2.17-SNAPSHOT
externalDocs:
  description: WFS specification
  url: 'https://github.com/opengeospatial/WFS_FES'
servers:
  - url: 'http://ows.geo-solutions.it/geoserver/ogc/features'
    description: This server
tags:
  - name: Capabilities
    description: essential characteristics of this API
  - name: Data
    description: access to data (features)
paths:
  /:
    get:
      tags:
        - Capabilities
      summary: landing page
      description: |-
        The landing page provides links to the API definition, the conformance
        statements and to the feature collections in this dataset.
      operationId: getLandingPage
      responses:
        '200':
          content:
            application/x-yaml:
              schema:
                type: string
                format: binary
            application/json:
              schema:
                type: string
                format: binary
            application/cbor:
              schema:
                type: string
                format: binary
            text/html:
              schema:
                type: string
          $ref: '#/components/responses/LandingPage'
        '500':
          $ref: '#/components/responses/ServerError'
  /conformance:
    get:
      tags:
        - Capabilities
      summary: information about specifications that this API conforms to
      description: |-
        A list of all conformance classes specified in a standard that the
        server conforms to.
      operationId: getConformanceDeclaration
      responses:
        '200':
          content:
            application/x-yaml:
              schema:
                type: string
                format: binary
            application/json:
              schema:
                type: string
                format: binary
            application/cbor:
              schema:
                type: string
                format: binary
            text/html:
              schema:
                type: string
          $ref: '#/components/responses/ConformanceDeclaration'
        '500':
          $ref: '#/components/responses/ServerError'
  /filter-capabilities:
    get:
      tags:
        - Capabilities
      summary: information about filters supported in the CQL filter extension
      description: A list of supported filters and functions.
      operationId: getFilterCapabilities
      responses:
        '200':
          $ref: '#/components/responses/FilterCapabilities'
        '500':
          $ref: '#/components/responses/ServerError'
  /collections:
    get:
      tags:
        - Capabilities
      summary: the feature collections in the dataset
      operationId: getCollections
      responses:
        '200':
          content:
            application/x-yaml:
              schema:
                type: string
                format: binary
            application/json:
              schema:
                type: string
                format: binary
            application/cbor:
              schema:
                type: string
                format: binary
            text/html:
              schema:
                type: string
          $ref: '#/components/responses/Collections'
        '500':
          $ref: '#/components/responses/ServerError'
  '/collections/{collectionId}':
    get:
      tags:
        - Capabilities
      summary: describe the feature collection with id `collectionId`
      operationId: describeCollection
      parameters:
        - $ref: '#/components/parameters/collectionId'
      responses:
        '200':
          content:
            application/x-yaml:
              schema:
                type: string
                format: binary
            application/json:
              schema:
                type: string
                format: binary
            application/cbor:
              schema:
                type: string
                format: binary
            text/html:
              schema:
                type: string
          $ref: '#/components/responses/Collection'
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/ServerError'
  '/collections/{collectionId}/queryables':
    get:
      tags:
        - Capabilities
      summary: >-
        lists the queryable attributes for the feature collection with id
        `collectionId`
      operationId: getQueryables
      parameters:
        - $ref: '#/components/parameters/collectionId'
      responses:
        '200':
          $ref: '#/components/responses/Queryables'
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/ServerError'
  '/collections/{collectionId}/items':
    get:
      tags:
        - Data
      summary: fetch features
      description: |-
        Fetch features of the feature collection with id `collectionId`.

        Every feature in a dataset belongs to a collection. A dataset may
        consist of multiple feature collections. A feature collection is often a
        collection of features of a similar type, based on a common schema.

        Use content negotiation to request HTML or GeoJSON.
      operationId: getFeatures
      parameters:
        - $ref: '#/components/parameters/collectionId'
        - $ref: '#/components/parameters/limit'
        - $ref: '#/components/parameters/bbox'
        - $ref: '#/components/parameters/datetime'
        - $ref: '#/components/parameters/filter'
        - $ref: '#/components/parameters/filter-lang'
      responses:
        '200':
          content:
            text/html:
              schema:
                type: string
            application/vnd.google-earth.kml+xml:
              schema:
                type: string
                format: binary
            application/geo+json:
              schema:
                type: string
                format: binary
            application/stac+json:
              schema:
                type: string
                format: binary
            application/gml+xml;version=3.2:
              schema:
                type: string
                format: binary
            application/json:
              schema:
                type: string
                format: binary
            application/cbor:
              schema:
                type: string
                format: binary
          $ref: '#/components/responses/Features'
        '400':
          $ref: '#/components/responses/InvalidParameter'
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/ServerError'
  '/collections/{collectionId}/items/{featureId}':
    get:
      tags:
        - Data
      summary: fetch a single feature
      description: |-
        Fetch the feature with id `featureId` in the feature collection
        with id `collectionId`.

        Use content negotiation to request HTML or GeoJSON.
      operationId: getFeature
      parameters:
        - $ref: '#/components/parameters/collectionId'
        - $ref: '#/components/parameters/featureId'
      responses:
        '200':
          content:
            text/html:
              schema:
                type: string
            application/vnd.google-earth.kml+xml:
              schema:
                type: string
                format: binary
            application/geo+json:
              schema:
                type: string
                format: binary
            application/stac+json:
              schema:
                type: string
                format: binary
            application/gml+xml;version=3.2:
              schema:
                type: string
                format: binary
            application/json:
              schema:
                type: string
                format: binary
            application/cbor:
              schema:
                type: string
                format: binary
          $ref: '#/components/responses/Feature'
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/ServerError'
components:
  schemas:
    queryables:
      required:
        - queryables
      type: object
      properties:
        queryables:
          type: array
          description: list of queryable properties
          items:
            $ref: '#/components/schemas/queryable'
    queryable:
      required:
        - name
        - type
      type: object
      properties:
        id:
          type: string
          description: identifier of the attribute that can be used in CQL filters
          example: address
        type:
          type: string
          description: the property type
          enum:
            - string
            - uri
          - number
          - integer
          - date
          - dateTime
          - boolean
          - geometry
    collection:
      required:
        - id
        - links
      type: object
      properties:
        id:
          type: string
          description: 'identifier of the collection used, for example, in URIs'
          example: address
        title:
          type: string
          description: human readable title of the collection
          example: address
        description:
          type: string
          description: a description of the features in the collection
          example: An address.
        links:
          type: array
          example:
            - href: 'http://data.example.com/buildings'
              rel: item
            - href: 'http://example.com/concepts/buildings.html'
              rel: describedBy
              type: text/html
          items:
            $ref: '#/components/schemas/link'
        extent:
          $ref: '#/components/schemas/extent'
        itemType:
          type: string
          description: >-
            indicator about the type of the items in the collection (the default
            value is 'feature').
          default: feature
        crs:
          type: array
          description: the list of coordinate reference systems supported by the
service
          example:
            - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
            - 'http://www.opengis.net/def/crs/EPSG/0/4326'
          items:
            type: string
          default:
            - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
    collections:
      required:
        - collections
        - links
      type: object
      properties:
        links:
          type: array
          items:
            $ref: '#/components/schemas/link'
        collections:
          type: array
          items:
            $ref: '#/components/schemas/collection'
    confClasses:
      required:
        - conformsTo
      type: object
      properties:
        conformsTo:
          type: array
          items:
            type: string
    exception:
      required:
        - code
      type: object
      properties:
        code:
          type: string
        description:
          type: string
      description: >-
        Information about the exception: an error code plus an optional
        description.
    extent:
      type: object
      properties:
        spatial:
          type: object
          properties:
            bbox:
              minItems: 1
              type: array
              description: >-
                One or more bounding boxes that describe the spatial extent of
                the dataset.

                In the Core only a single bounding box is supported. Extensions
                may support

                additional areas. If multiple areas are provided, the union of
                the bounding

                boxes describes the spatial extent.
              items:
                maxItems: 6
                minItems: 4
                type: array
                description: >-
                  Each bounding box is provided as four or six numbers,
                  depending on

                  whether the coordinate reference system includes a vertical
                  axis

                  (height or depth):


                  * Lower left corner, coordinate axis 1

                  * Lower left corner, coordinate axis 2

                  * Minimum value, coordinate axis 3 (optional)

                  * Upper right corner, coordinate axis 1

                  * Upper right corner, coordinate axis 2

                  * Maximum value, coordinate axis 3 (optional)


                  The coordinate reference system of the values is WGS 84
                  longitude/latitude

                  (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a
                  different coordinate

                  reference system is specified in `crs`.


                  For WGS 84 longitude/latitude the values are in most cases the
                  sequence of

                  minimum longitude, minimum latitude, maximum longitude and
                  maximum latitude.

                  However, in cases where the box spans the antimeridian the
                  first value
                  (west-most box edge) is larger than the third value (east-most
                  box edge).


                  If the vertical axis is included, the third and the sixth
                  number are

                  the bottom and the top of the 3-dimensional bounding box.


                  If a feature has multiple spatial geometry properties, it is
                  the decision of the

                  server whether only a single spatial geometry property is used
                  to determine

                  the extent or all relevant geometries.
                example:
                  - -180
                  - -90
                  - 180
                  - 90
                items:
                  type: number
            crs:
              type: string
              description: >-
                Coordinate reference system of the coordinates in the spatial
                extent

                (property `bbox`). The default reference system is WGS 84
                longitude/latitude.

                In the Core this is the only supported coordinate reference
                system.

                Extensions may support additional coordinate reference systems
                and add

                additional enum values.
              enum:
                - 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
              default: 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'
          description: The spatial extent of the features in the collection.
        temporal:
          type: object
          properties:
            interval:
              minItems: 1
              type: array
              description: >-
```

```
          One or more time intervals that describe the temporal extent of
          the dataset.

          The value `null` is supported and indicates an open time
          intervall.

          In the Core only a single time interval is supported. Extensions
          may support

          multiple intervals. If multiple intervals are provided, the
          union of the

          intervals describes the temporal extent.
        items:
          maxItems: 2
          minItems: 2
          type: array
          description: >-
            Begin and end times of the time interval. The timestamps

            are in the coordinate reference system specified in `trs`. By
            default

            this is the Gregorian calendar.
          example:
            - '2011-11-11T12:22:11Z'
            - null
          items:
            type: string
            format: date-time
            nullable: true
      trs:
        type: string
        description: >-
          Coordinate reference system of the coordinates in the temporal
          extent

          (property `interval`). The default reference system is the
          Gregorian calendar.

          In the Core this is the only supported temporal reference
          system.

          Extensions may support additional temporal reference systems and
          add

          additional enum values.
        enum:
          - 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
        default: 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'
    description: The temporal extent of the features in the collection.
```

```
      description: >-
        The extent of the features in the collection. In the Core only spatial
        and temporal

        extents are specified. Extensions may add additional members to
        represent other

        extents, for example, thermal or pressure ranges.
    featureCollectionGeoJSON:
      required:
        - features
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - FeatureCollection
        features:
          type: array
          items:
            $ref: '#/components/schemas/featureGeoJSON'
        links:
          type: array
          items:
            $ref: '#/components/schemas/link'
        timeStamp:
          $ref: '#/components/schemas/timeStamp'
        numberMatched:
          $ref: '#/components/schemas/numberMatched'
        numberReturned:
          $ref: '#/components/schemas/numberReturned'
    featureGeoJSON:
      required:
        - geometry
        - properties
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - Feature
        geometry:
          $ref: '#/components/schemas/geometryGeoJSON'
        properties:
          type: object
          nullable: true
        id:
          oneOf:
            - type: string
          - type: integer
        links:
          type: array
          items:
            $ref: '#/components/schemas/link'
    geometryGeoJSON:
      oneOf:
        - $ref: '#/components/schemas/pointGeoJSON'
        - $ref: '#/components/schemas/multipointGeoJSON'
        - $ref: '#/components/schemas/linestringGeoJSON'
        - $ref: '#/components/schemas/multilinestringGeoJSON'
        - $ref: '#/components/schemas/polygonGeoJSON'
        - $ref: '#/components/schemas/multipolygonGeoJSON'
        - $ref: '#/components/schemas/geometrycollectionGeoJSON'
    geometrycollectionGeoJSON:
      required:
        - geometries
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - GeometryCollection
        geometries:
          type: array
          items:
            $ref: '#/components/schemas/geometryGeoJSON'
    landingPage:
      required:
        - links
      type: object
      properties:
        title:
          type: string
          example: Buildings in Bonn
        description:
          type: string
          example: >-
            Access to data about buildings in the city of Bonn via a Web API
            that conforms to the OGC API Features specification.
        links:
          type: array
          items:
            $ref: '#/components/schemas/link'
    linestringGeoJSON:
      required:
        - coordinates
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - LineString
        coordinates:
          minItems: 2
          type: array
          items:
            minItems: 2
            type: array
            items:
              type: number
    link:
      required:
        - href
      type: object
      properties:
        href:
          type: string
          example: 'http://data.example.com/buildings/123'
        rel:
          type: string
          example: alternate
        type:
          type: string
          example: application/geo+json
        hreflang:
          type: string
          example: en
        title:
          type: string
          example: 'Trierer Strasse 70, 53115 Bonn'
        length:
          type: integer
    multilinestringGeoJSON:
      required:
        - coordinates
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - MultiLineString
        coordinates:
          type: array
          items:
            minItems: 2
            type: array
            items:
              minItems: 2
          type: array
          items:
            type: number
    multipointGeoJSON:
      required:
        - coordinates
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - MultiPoint
        coordinates:
          type: array
          items:
            minItems: 2
            type: array
            items:
              type: number
    multipolygonGeoJSON:
      required:
        - coordinates
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - MultiPolygon
        coordinates:
          type: array
          items:
            type: array
            items:
              minItems: 4
              type: array
              items:
                minItems: 2
                type: array
                items:
                  type: number
    numberMatched:
      minimum: 0
      type: integer
      description: |-
        The number of features of the feature type that match the selection
        parameters like `bbox`.
      example: 127
    numberReturned:
      minimum: 0
      type: integer
      description: |-
        The number of features in the feature collection.

        A server may omit this information in a response, if the information
        about the number of features is not known or difficult to compute.

        If the value is provided, the value shall be identical to the number
        of items in the "features" array.
      example: 10
    pointGeoJSON:
      required:
        - coordinates
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - Point
        coordinates:
          minItems: 2
          type: array
          items:
            type: number
    polygonGeoJSON:
      required:
        - coordinates
        - type
      type: object
      properties:
        type:
          type: string
          enum:
            - Polygon
        coordinates:
          type: array
          items:
            minItems: 4
            type: array
            items:
              minItems: 2
              type: array
              items:
                type: number
    timeStamp:
      type: string
      description: >-
        This property indicates the time and date when the response was
        generated.
      format: date-time
```

```
  responses:
    LandingPage:
      description: |-
        The landing page provides links to the API definition
        (link relations `service-desc` and `service-doc`),
        the Conformance declaration (path `/conformance`,
        link relation `conformance`), and the Feature
        Collections (path `/collections`, link relation
        `data`).
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/landingPage'
          example:
            title: Buildings in Bonn
            description: >-
              Access to data about buildings in the city of Bonn via a Web API
              that conforms to the OGC API Features specification.
            links:
              - href: 'http://data.example.org/'
                rel: self
                type: application/json
                title: this document
              - href: 'http://data.example.org/api'
                rel: service-desc
                type: application/vnd.oai.openapi+json;version=3.0
                title: the API definition
              - href: 'http://data.example.org/api.html'
                rel: service-doc
                type: text/html
                title: the API documentation
              - href: 'http://data.example.org/conformance'
                rel: conformance
                type: application/json
                title: OGC API conformance classes implemented by this server
              - href: 'http://data.example.org/collections'
                rel: data
                type: application/json
                title: Information about the feature collections
        text/html:
          schema:
            type: string
    ConformanceDeclaration:
      description: |-
        The URIs of all conformance classes supported by the server.

        To support "generic" clients that want to access multiple
        OGC API Features implementations - and not "just" a specific
        API / server, the server declares the conformance
        classes it implements and conforms to.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/confClasses'
          example:
            conformsTo:
              - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core'
              - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/oas30'
              - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/html'
              - 'http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/geojson'
        text/html:
          schema:
            type: string
    Collections:
      description: >-
        The feature collections shared by this API.


        The dataset is organized as one or more feature collections. This
        resource

        provides information about and access to the collections.


        The response contains the list of collections. For each collection, a
        link

        to the items in the collection (path
        `/collections/{collectionId}/items`,

        link relation `items`) as well as key information about the collection.

        This information includes:


        * A local identifier for the collection that is unique for the dataset;

        * A list of coordinate reference systems (CRS) in which geometries may
        be returned by the server. The first CRS is the default coordinate
        reference system (the default is always WGS 84 with axis order
        longitude/latitude);

        * An optional title and description for the collection;

        * An optional extent that can be used to provide an indication of the
        spatial and temporal extent of the collection - typically derived from
        the data;

        * An optional indicator about the type of the items in the collection
        (the default value, if the indicator is not provided, is 'feature').
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/collections'
          example:
            links:
              - href: 'http://data.example.org/collections.json'
                rel: self
                type: application/json
                title: this document
              - href: 'http://data.example.org/collections.html'
                rel: alternate
                type: text/html
                title: this document as HTML
              - href: 'http://schemas.example.org/1.0/buildings.xsd'
                rel: describedBy
                type: application/xml
                title: GML application schema for Acme Corporation building data
              - href: 'http://download.example.org/buildings.gpkg'
                rel: enclosure
                type: application/geopackage+sqlite3
                title: Bulk download (GeoPackage)
                length: 472546
            collections:
              - id: buildings
                title: Buildings
                description: Buildings in the city of Bonn.
                extent:
                  spatial:
                    bbox:
                      - - 7.01
                        - 50.63
                        - 7.22
                        - 50.78
                  temporal:
                    interval:
                      - - '2010-02-15T12:34:56Z'
                        - null
                links:
                  - href: 'http://data.example.org/collections/buildings/items'
                    rel: items
                    type: application/geo+json
                    title: Buildings
                  - href: 'http://data.example.org/collections/buildings/items.html'
                    rel: items
                    type: text/html
                    title: Buildings
                  - href: 'https://creativecommons.org/publicdomain/zero/1.0/'
                    rel: license
                    type: text/html
                    title: CC0-1.0
                  - href: 'https://creativecommons.org/publicdomain/zero/1.0/rdf'
                    rel: license
                    type: application/rdf+xml
                    title: CC0-1.0
        text/html:
          schema:
            type: string
    Queryables:
      description: Information about the feature collection queryable properties
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/queryables'
    Collection:
      description: >-
        Information about the feature collection with id `collectionId`.


        The response contains a link to the items in the collection

        (path `/collections/{collectionId}/items`, link relation `items`)

        as well as key information about the collection. This information

        includes:


        * A local identifier for the collection that is unique for the dataset;

        * A list of coordinate reference systems (CRS) in which geometries may
        be returned by the server. The first CRS is the default coordinate
        reference system (the default is always WGS 84 with axis order
        longitude/latitude);

        * An optional title and description for the collection;

        * An optional extent that can be used to provide an indication of the
        spatial and temporal extent of the collection - typically derived from
        the data;

        * An optional indicator about the type of the items in the collection
        (the default value, if the indicator is not provided, is 'feature').
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/collection'
          example:
            id: buildings
            title: Buildings
            description: Buildings in the city of Bonn.
            extent:
              spatial:
                bbox:
```

```
              - - 7.01
                - 50.63
                - 7.22
                - 50.78
          temporal:
            interval:
              - - '2010-02-15T12:34:56Z'
                - null
        links:
          - href: 'http://data.example.org/collections/buildings/items'
            rel: items
            type: application/geo+json
            title: Buildings
          - href: 'http://data.example.org/collections/buildings/items.html'
            rel: items
            type: text/html
            title: Buildings
          - href: 'https://creativecommons.org/publicdomain/zero/1.0/'
            rel: license
            type: text/html
            title: CC0-1.0
          - href: 'https://creativecommons.org/publicdomain/zero/1.0/rdf'
            rel: license
            type: application/rdf+xml
            title: CC0-1.0
    text/html:
      schema:
        type: string
Features:
  description: >-
    The response is a document consisting of features in the collection.

    The features included in the response are determined by the server

    based on the query parameters of the request. To support access to

    larger collections without overloading the client, the API supports

    paged access with links to the next page, if more features are selected

    that the page size.


    The `bbox` and `datetime` parameter can be used to select only a

    subset of the features in the collection (the features that are in the

    bounding box or time interval). The `bbox` parameter matches all
    features

    in the collection that are not associated with a location, too. The
```

```
        `datetime` parameter matches all features in the collection that are

        not associated with a time stamp or interval, too.


        The `limit` parameter may be used to control the subset of the

        selected features that should be returned in the response, the page
        size.

        Each page may include information about the number of selected and

        returned features (`numberMatched` and `numberReturned`) as well as

        links to support paging (link relation `next`).
      content:
        application/geo+json:
          schema:
            $ref: '#/components/schemas/featureCollectionGeoJSON'
          example:
            type: FeatureCollection
            links:
              - href: 'http://data.example.com/collections/buildings/items.json'
                rel: self
                type: application/geo+json
                title: this document
              - href: 'http://data.example.com/collections/buildings/items.html'
                rel: alternate
                type: text/html
                title: this document as HTML
              - href: >-

http://data.example.com/collections/buildings/items.json&offset=10&limit=2
                rel: next
                type: application/geo+json
                title: next page
            timeStamp: '2018-04-03T14:52:23Z'
            numberMatched: 123
            numberReturned: 2
            features:
              - type: Feature
                id: '123'
                geometry:
                  type: Polygon
                  coordinates:
                    - ...
                properties:
                  function: residential
                  floors: '2'
                  lastUpdate: '2015-08-01T12:34:56Z'
              - type: Feature
                id: '132'
                geometry:
                  type: Polygon
                  coordinates:
                    - ...
                properties:
                  function: public use
                  floors: '10'
                  lastUpdate: '2013-12-03T10:15:37Z'
        text/html:
          schema:
            type: string
    Feature:
      description: |-
        fetch the feature with id `featureId` in the feature collection
        with id `collectionId`
      content:
        application/geo+json:
          schema:
            $ref: '#/components/schemas/featureGeoJSON'
          example:
            type: Feature
            links:
              - href: 'http://data.example.com/id/building/123'
                rel: canonical
                title: canonical URI of the building
              - href: 'http://data.example.com/collections/buildings/items/123.json'
                rel: self
                type: application/geo+json
                title: this document
              - href: 'http://data.example.com/collections/buildings/items/123.html'
                rel: alternate
                type: text/html
                title: this document as HTML
              - href: 'http://data.example.com/collections/buildings'
                rel: collection
                type: application/geo+json
                title: the collection document
            id: '123'
            geometry:
              type: Polygon
              coordinates:
                - ...
            properties:
              function: residential
              floors: '2'
              lastUpdate: '2015-08-01T12:34:56Z'
        text/html:
          schema:
            type: string
    InvalidParameter:
      description: A query parameter has an invalid value.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/exception'
        text/html:
          schema:
            type: string
    NotFound:
      description: The requested URI was not found.
    ServerError:
      description: A server error occurred.
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/exception'
        text/html:
          schema:
            type: string
    FilterCapabilities:
      description: A document listing the server filtering capabilities
      content:
        application/json:
          schema:
            type: object
        text/html:
          schema:
            type: string
  parameters:
    bbox:
      name: bbox
      in: query
      description: >-
        Only features that have a geometry that intersects the bounding box are
        selected.

        The bounding box is provided as four or six numbers, depending on
        whether the

        coordinate reference system includes a vertical axis (height or depth):


        * Lower left corner, coordinate axis 1

        * Lower left corner, coordinate axis 2

        * Minimum value, coordinate axis 3 (optional)

        * Upper right corner, coordinate axis 1
        * Upper right corner, coordinate axis 2

        * Maximum value, coordinate axis 3 (optional)


        The coordinate reference system of the values is WGS 84
        longitude/latitude

        (http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different
        coordinate

        reference system is specified in the parameter `bbox-crs`.


        For WGS 84 longitude/latitude the values are in most cases the sequence
        of

        minimum longitude, minimum latitude, maximum longitude and maximum
        latitude.

        However, in cases where the box spans the antimeridian the first value

        (west-most box edge) is larger than the third value (east-most box
        edge).


        If the vertical axis is included, the third and the sixth number are

        the bottom and the top of the 3-dimensional bounding box.


        If a feature has multiple spatial geometry properties, it is the
        decision of the

        server whether only a single spatial geometry property is used to
        determine

        the extent or all relevant geometries.
      required: false
      style: form
      explode: false
      schema:
        maxItems: 6
        minItems: 4
        type: array
        items:
          type: number
    collectionId:
      name: collectionId
      in: path
      description: local identifier of a collection
```

```
        required: true
        schema:
          type: string
          enum:
            - 'syria_vtp:building_s'
            - 'syria_vtp:built_up_area_s'
            - 'syria_vtp:cemetery_s'
            - 'syria_vtp:crop_land_s'
            - 'syria_vtp:dam_s'
            - 'syria_vtp:electric_power_station_s'
            - 'syria_vtp:facility_s'
            - 'syria_vtp:grassland_s'
            - 'syria_vtp:military_installation_s'
            - 'syria_vtp:power_substation_s'
            - 'syria_vtp:river_c'
            - 'syria_vtp:river_s'
            - 'syria_vtp:road_c'
            - 'syria_vtp:settlement_s'
            - 'syria_vtp:tower_s'
            - 'vtp:AeronauticPnt'
            - 'vtp:AgriculturePnt'
            - 'vtp:AgricultureSrf'
            - 'vtp:CulturePnt'
            - 'vtp:CultureSrf'
            - 'vtp:FacilityPnt'
            - 'vtp:HydrographyCrv'
            - 'vtp:HydrographyPnt'
            - 'vtp:HydrographySrf'
            - 'vtp:MilitarySrf'
            - 'vtp:SettlementSrf'
            - 'vtp:StoragePnt'
            - 'vtp:StructurePnt'
            - 'vtp:TransportationGroundCrv'
            - 'vtp:UtilityInfrastructureCrv'
            - 'vtp:UtilityInfrastructurePnt'
            - 'vtp:VegetationSrf'
            - 'iraq_vtp:aircraft_hangar_s'
            - 'iraq_vtp:amusement_park_s'
            - 'iraq_vtp:annotated_location_s'
            - 'iraq_vtp:apron_s'
            - 'iraq_vtp:archeological_site_s'
            - 'iraq_vtp:barn_s'
            - 'iraq_vtp:bridge_c'
            - 'iraq_vtp:bridge_s'
            - 'iraq_vtp:brush_s'
            - 'iraq_vtp:building_p'
            - 'iraq_vtp:building_s'
            - 'iraq_vtp:built_up_area_p'
            - 'iraq_vtp:built_up_area_s'
            - 'iraq_vtp:canal_c'
            - 'iraq_vtp:cart_track_c'
            - 'iraq_vtp:castle_s'
            - 'iraq_vtp:cemetery_s'
            - 'iraq_vtp:crop_land_s'
            - 'iraq_vtp:crossing_p'
            - 'iraq_vtp:culvert_c'
            - 'iraq_vtp:cut_c'
            - 'iraq_vtp:dam_c'
            - 'iraq_vtp:dam_s'
            - 'iraq_vtp:disposal_site_s'
            - 'iraq_vtp:ditch_c'
            - 'iraq_vtp:embankment_c'
            - 'iraq_vtp:extraction_mine_s'
            - 'iraq_vtp:facility_s'
            - 'iraq_vtp:fence_c'
            - 'iraq_vtp:firing_range_s'
            - 'iraq_vtp:ford_c'
            - 'iraq_vtp:forest_s'
            - 'iraq_vtp:fountain_s'
            - 'iraq_vtp:gate_c'
            - 'iraq_vtp:gate_p'
            - 'iraq_vtp:grain_storage_structure_s'
            - 'iraq_vtp:grassland_s'
            - 'iraq_vtp:greenhouse_s'
            - 'iraq_vtp:helipad_p'
            - 'iraq_vtp:helipad_s'
            - 'iraq_vtp:hut_s'
            - 'iraq_vtp:hydrocarbons_field_s'
            - 'iraq_vtp:inland_waterbody_s'
            - 'iraq_vtp:interest_site_p'
            - 'iraq_vtp:interest_site_s'
            - 'iraq_vtp:island_s'
            - 'iraq_vtp:land_aerodrome_p'
            - 'iraq_vtp:land_aerodrome_s'
            - 'iraq_vtp:land_water_boundary_c'
            - 'iraq_vtp:lookout_s'
            - 'iraq_vtp:marsh_s'
            - 'iraq_vtp:memorial_monument_s'
            - 'iraq_vtp:military_installation_s'
            - 'iraq_vtp:motor_vehicle_station_p'
            - 'iraq_vtp:motor_vehicle_station_s'
            - 'iraq_vtp:orchard_s'
            - 'iraq_vtp:park_s'
            - 'iraq_vtp:power_substation_s'
            - 'iraq_vtp:racetrack_c'
            - 'iraq_vtp:racetrack_s'
            - 'iraq_vtp:railway_c'
            - 'iraq_vtp:railway_sidetrack_c'
            - 'iraq_vtp:river_c'
            - 'iraq_vtp:river_s'
            - 'iraq_vtp:road_c'
            - 'iraq_vtp:road_s'
```

```
      - 'iraq_vtp:roadside_rest_area_s'
      - 'iraq_vtp:ruins_p'
      - 'iraq_vtp:ruins_s'
      - 'iraq_vtp:settlement_s'
      - 'iraq_vtp:sewage_treatment_plant_s'
      - 'iraq_vtp:shed_s'
      - 'iraq_vtp:shopping_complex_s'
      - 'iraq_vtp:sports_ground_s'
      - 'iraq_vtp:stadium_s'
      - 'iraq_vtp:stair_c'
      - 'iraq_vtp:steep_terrain_face_c'
      - 'iraq_vtp:storage_tank_s'
      - 'iraq_vtp:swamp_s'
      - 'iraq_vtp:swimming_pool_s'
      - 'iraq_vtp:taxiway_c'
      - 'iraq_vtp:tower_s'
      - 'iraq_vtp:traffic_light_p'
      - 'iraq_vtp:trail_c'
      - 'iraq_vtp:transportation_block_p'
      - 'iraq_vtp:transportation_station_s'
      - 'iraq_vtp:tunnel_c'
      - 'iraq_vtp:vehicle_barrier_c'
      - 'iraq_vtp:vehicle_lot_s'
      - 'iraq_vtp:wall_c'
      - 'iraq_vtp:water_tower_s'
      - 'iraq_vtp:water_well_s'
      - 'iraq_vtp:waterwork_s'
      - 'iraq_vtp:zoo_s'
      - 'ne:countries50m'
      - 'ne:popplaces50m'
      - 'ne:urban50m'
  datetime:
    name: datetime
    in: query
    description: >-
      Either a date-time or an interval, open or closed. Date and time
      expressions

      adhere to RFC 3339. Open intervals are expressed using double-dots.


      Examples:


      * A date-time: "2018-02-12T23:20:50Z"

      * A closed interval: "2018-02-12T00:00:00Z/2018-03-18T12:31:12Z"

      * Open intervals: "2018-02-12T00:00:00Z/.." or "../2018-03-18T12:31:12Z"

      Only features that have a temporal property that intersects the value of

      `datetime` are selected.


      If a feature has multiple temporal properties, it is the decision of the

      server whether only a single temporal property is used to determine

      the extent or all relevant temporal properties.
    required: false
    style: form
    explode: false
    schema:
      type: string
  featureId:
    name: featureId
    in: path
    description: local identifier of a feature
    required: true
    schema:
      type: string
  limit:
    name: limit
    in: query
    description: >-
      The optional limit parameter limits the number of items that are
      presented in the response document.


      Only items are counted that are on the first level of the collection in
      the response document.

      Nested objects contained within the explicitly requested items shall not
      be counted.


      Minimum = 1. Maximum = 10000. Default = 10.
    required: false
    style: form
    explode: false
    schema:
      maximum: 1000000
      minimum: 1
      type: integer
      default: 1000000
  filter:
    name: filter
    in: query
    description: >-
      Defines a filter that will be applied on items, only items matching the
```

```
        filter will be returned
      schema:
        type: string
    filter-lang:
      name: filter-lang
      in: query
      description: Filter encoding used in the filter parameter
      schema:
        type: string
        enum:
          - cql-text
        default: cql-text
```

Appendix B: OGC API Features Part1:Core output (JSON)

```
{
   "openapi":"3.0.2",
   "externalDocs":{
      "description":"WFS Specification",
      "url":"https://github.com/opengeospatial/WFS_FES"
   },
   "servers":[
      {
         "url":"http://ows.geo-solutions.it/geoserver/ogc/features",
         "description":"This server"
      }
   ],
   "info":{
      "title":"Features 1.0 server",
      "version":"2.17-SNAPSHOT",
      "contact":{
         "name":""
      }
   },
   "tags":[
      {
         "name":"Data",
         "description":"access to data (features)"
      },
      {
         "name":"Capabilities",
         "description":"essential characteristics of this API"
      }
   ],
   "paths":{
      "/":{
         "get":{
            "summary":"landing page",
            "description":"The landing page provides links to the API definition, the
conformance\n        statements and to the feature collections in this dataset.",
            "operationId":"getLandingPage",
            "tags":[
               "Capabilities"
            ],
            "responses":{
               200:{
                  "description":"200 response",
                  "content":{
                     "application/x-yaml":{
                        "schema":{
```

```
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/cbor":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "text/html":{
                     "schema":{
                        "type":"string"
                     }
                  }
               },
               "$ref":{
                  "$ref":"#/components/schemas/landingPage"
               }
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            }
         }
      }
   },
   "/conformance":{
      "get":{
         "summary":"information about specifications that this API conforms to",
         "description":"A list of all conformance classes specified in a standard
the server conforms to.",
         "operationId":"getConformanceDeclaration",
         "responses":{
            200:{
               "description":"200 response",
               "content":{
                  "application/x-yaml":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/cbor":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "text/html":{
                     "schema":{
                        "type":"string"
                     }
                  }
               },
               "$ref":{
                  "$ref":"#/components/schemas/landingPage"
               }
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            }
         },
         "tags":[
            "Capabilities"
         ]
      }
   },
   "/filter-capabilties":{
      "get":{
         "summary":"information about filters supported in the CQL filter
extension",
         "description":"A list of",
         "operationId":"getFilterCapabilities",
         "responses":{
            200:{
               "$ref":"#/components/responses/FilterCapabilities"
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            }
         },
         "tags":[
            "Capabilities"
         ]
      }
   },
   "/collections":{
      "get":{
         "summary":"describe the feature collection with id `collectionId`",
         "operationId":"getCollections",
         "responses":{
            200:{
               "description":"200 response",
               "content":{
                  "application/x-yaml":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/cbor":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "text/html":{
                     "schema":{
                        "type":"string"
                     }
                  }
               },
               "$ref":{
                  "$ref":"#/components/schemas/landingPage"
               }
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            }
         },
         "tags":[
            "Capabilities"
         ]
      }
   },
   "/collections/{collectionId}":{
      "get":{
         "summary":"describe the feature collection with id 'collectionId'",
         "operationId":"describeCollection",
         "parameters":[
            {
               "$ref":"#/components/parameters/collectionId"
            }
         ],
         "responses":{
            200:{
               "description":"200 response",
               "content":{
                  "application/x-yaml":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/cbor":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "text/html":{
                     "schema":{
                        "type":"string"
                     }
                  }
               },
               "$ref":{
                  "$ref":"#/components/schemas/landingPage"
               }
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            }
         },
         "tags":[
            "Capabilities"
         ]
      }
   },
   "/collections/{collectionId}/queryables":{
      "get":{
         "summary":"lists the querable attributes for the feature collection with
id 'collectionId'",
         "operationId":"getQueryables",
         "parameters":[
            {
               "$ref":"#/components/parameters/collectionId"
            }
         ],
         "responses":{
```

```
            404:{
               "$ref":"#/components/responses/NotFound"
            },
            200:{
               "$ref":"#/components/responses/Queryables"
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            }
         },
         "tags":[
            "Capabilities"
         ]
      }
   },
   "/collections/{collectionId}/items":{
      "get":{
         "summary":"fetch features",
         "description":"Fetch features of the feature collection with id
`collectionId`.\n\n        Every feature in a dataset belongs to a collection. A
dataset may\n        consist of multiple feature collections. A feature collection is
often a\n        collection of features of a similar type, based on a common
schema.\n\n        Use content negotiation to request HTML or GeoJSON.",
         "operationId":"getFeatures",
         "parameters":[
            {
               "$ref":"#/components/parameters/collectionId"
            },
            {
               "$ref":"#/components/parameters/filter"
            },
            {
               "$ref":"#/components/parameters/limit"
            },
            {
               "$ref":"#/components/parameters/bbox"
            },
            {
               "$ref":"#/components/parameters/datetime"
            },
            {
               "$ref":"#/components/parameters/filter-lang"
            }
         ],
         "responses":{
            404:{
               "$ref":"#/components/responses/NotFound"
            },
            500:{
               "$ref":"#/components/responses/ServerError"
            },
            400:{
               "$ref":"#/components/responses/InvalidParameter"
            },
            200:{
               "content":{
                  "application/cbor":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/vnd.google-earth.kml+xml":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/geo+json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/stac+json":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  },
                  "application/gml+xml;version=3.2":{
                     "schema":{
                        "type":"string",
                        "format":"binary"
                     }
                  }
               }
            }
         },
         "tags":[
            "- Data"
         ]
      }
   }
},
"components":{
      "schemas":{
         "queryables":{
            "type":"object",
            "properties":{
               "Schema queryables Property":{
                  "type":"array",
                  "description":"list of queryable properties"
               }
            }
         },
         "queryable":{
            "type":"object",
            "required":[
               "- name",
               "- type"
            ],
            "properties":{
               "Schema queryable id Property":{
                  "type":"string",
                  "description":"identifier of the attribute that can be used in CQL
filters",
                  "example":"address"
               },
               "type":{
                  "type":"string",
                  "description":"the property type",
                  "enum":[
                     "- dateTime",
                     "- boolean",
                     "- date",
                     "- number",
                     "- geometry",
                     "- integer",
                     "- string",
                     "- url"
                  ]
               }
            }
         },
         "collection":{
            "type":"object",
            "required":[
               "- id",
               "- links"
            ],
            "properties":{
               "Schema extent Reference":{
                  "$ref":"#/components/schemas/extent"
               },
               "Schema collection id Properties":{
                  "type":"string",
                  "description":"identifier of teh collection used, for example, in
URIs",
                  "example":"address"
               },
               "Schema collection title Properties":{
                  "type":"string",
                  "description":"human readable title of the collection",
                  "example":"address"
               },
               "Schema collection description Properties":{
                  "type":"string",
                  "description":"a description of hte features in the collection",
                  "example":"An address."
               },
               "Schema collection links Properties":{
                  "type":"array",
                  "example":"href: 'http://data.example.com/buildings'\n
rel: item\n            - href: 'http://example.com/concepts/buildings.html'\n
rel: describedBy\n              type: text/html",
                  "items":{
                     "$ref":"#/components/schemas/link"
                  }
               },
               "Schema collection itemType Properties":{
                  "type":"String",
                  "description":"indicator about the type of the items in the
collection (the default\n            value is 'feature').",
                  "default":"feature"
               },
               "Schema collection crs Properties":{
                  "type":"array",
                  "description":"the list of coordinate reference systems supported by
the service",
                  "example":"- 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'\n
- 'http://www.opengis.net/def/crs/EPSG/0/4326'",
                  "default":"- 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'",
                  "items":{
                     "type":"string"
                  }
               }
            }
         },
         "collections":{
            "type":"object",
            "required":[
               "- collections",
               "- links"
            ],
            "properties":{
               "Schema collections links property":{
                  "type":"array",
                  "items":{
                     "$ref":"#/components/schemas/link"
                  }
               },
               "Schema collections collections property":{
                  "type":"array",
                  "items":{
                     "$ref":"#/components/schemas/collection"
                  }
               }
            }
         },
         "confClasses":{
            "type":"array",
            "required":[
               "conformsTo"
            ],
            "properties":{
               "Schema confClasses conformsTo property":{
                  "type":"array",
                  "items":{
                     "type":"String"
                  }
               }
            }
         },
         "exception":{
            "requried":[
               "- code",
               {
                  "name":"- code"
               }
            ],
            "type":"object",
            "description":">-\n        Information about the exception: an error code
plus an optional\n        description.",
            "properties":{
               "Schema exception code property":{
                  "type":"string"
               },
               "Schema exception description property":{
                  "type":"string"
               }
            }
         },
         "extent":{
            "type":"object",
            "description":">-\n        The extent of the features in the collection.
In the Core only spatial\n        and temporal\n\n        extents are specified.
Extensions may add additional members to\n        represent other\n\n        extents,
for example, thermal or pressure ranges.",
            "properties":{
               "Schema extent bbox spatial property":{
                  "minItems":1,
                  "type":"object",
                  "description":">-\n                One or more bounding boxes that
describe the spatial extent of\n                the dataset.\n\n                In the
Core only a single bounding box is supported. Extensions\n                may
support\n\n                additional areas. If multiple areas are provided, the union
of\n                the bounding\n\n                boxes describes the spatial
extent.",
                  "properties":[
                     {
                        "maxItems":6,
                        "minItems":4,
                        "type":"array",
                        "description":">-\n                  Each bounding box is
provided as four or six numbers,\n                  depending on\n\n
whether the coordinate reference system includes a vertical\n
axis\n\n                  (height or depth):\n\n\n                  * Lower left
corner, coordinate axis 1\n\n                  * Lower left corner, coordinate axis
2\n\n                  * Minimum value, coordinate axis 3 (optional)\n\n
* Upper right corner, coordinate axis 1\n\n                  * Upper right corner,
coordinate axis 2\n\n                  * Maximum value, coordinate axis 3
(optional)\n\n\n                  The coordinate reference system of the values is WGS
84\n                  longitude/latitude\n\n
(http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a\n                  different
coordinate\n\n                  reference system is specified in `crs`.\n\n\n
For WGS 84 longitude/latitude the values are in most cases the\n
sequence of\n\n                  minimum longitude, minimum latitude, maximum
longitude and\n                  maximum latitude.\n\n                  However, in
cases where the box spans the antimeridian the\n                  first value\n\n
(west-most box edge) is larger than the third value (east-most\n                  box
edge).\n\n\n                  If the vertical axis is included, the third and the
sixth\n                  number are\n\n                  the bottom and the top of the
3-dimensional bounding box.\n\n\n                  If a feature has multiple spatial
geometry properties, it is\n                  the decision of the\n\n
server whether only a single spatial geometry property is used\n                  to
determine\n\n                  the extent or all relevant geometries.",
                        "example":"- -180\n                  - -90\n
- 180\n                  - 90",
                        "items":{
                           "type":"number"
                        }
                     },
                     {
                        "type":"string",
                        "description":"Coordinate reference system of the coordinates
in the spatial\n                extent\n\n                (property `bbox`). The
default reference system is WGS 84\n                longitude/latitude.\n\n
In the Core this is the only supported coordinate reference\n
system.\n\n                Extensions may support additional coordinate reference
systems\n                and add\n\n                additional enum values.",
                        "enum":[
                           "- 'http://www.opengis.net/def/crs/OGC/1.3/CRS84'"
                        ],
                        "default":"http://www.opengis.net/def/crs/OGC/1.3/CRS84"
                     }
                  ]
               },
               "Schema extent bbox temporal property":{
                  "type":"object",
                  "properties":{
                     "minItems":1,
                     "type":"array",
                     "description":"One or more time intervals that describe the
temporal extent of\n                the dataset.\n\n                The value `null`
is supported and indicates an open time\n                intervall.\n\n
In the Core only a single time interval is supported. Extensions\n                may
support\n\n               multiple intervals. If multiple intervals are provided,
the\n                union of the\n\n                intervals describes the temporal
extent.",
                     "items":{
                        "maxItems":2,
                        "minItems":2,
                        "type":"array",
                        "description":"Begin and end times of the time interval. The
timestamps\n\n                 are in the coordinate reference system specified in
`trs`. By\n                  default\n\n                  this is the Gregorian
calendar.",
                        "example":"- '2011-11-11T12:22:11Z'\n                  -
null",
                        "items":{
                           "type":"String",
                           "format":"date-time",
                           "nullable":true
                        }
                     }
                  }
               },
               "Schema extent bbox temporal trs property":{
                  "type":"string",
                  "description":"Coordinate reference system of the coordinates in the
temporal\n                extent\n\n                (property `interval`). The default
reference system is the\n                Gregorian calendar.\n\n                In the
Core this is the only supported temporal reference\n                system.\n\n
Extensions may support additional temporal reference systems and\n
add\n\n                additional enum values.",
                  "enum":[
                     "- 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'"
                  ],
                  "default":"- 'http://www.opengis.net/def/uom/ISO-8601/0/Gregorian'"
               }
```

```
         }
      },
      "featureCollectionGeoJSON":{
         "type":"object",
         "required":[
            "- features",
            "- type"
         ],
         "properties":{
            "type":{
               "enum":[
                  "- featureCollection"
               ]
            },
            "Schema featureCollectionGeoJSON features property":{
               "type":"array",
               "items":{
                  "$ref":"#/components/schemas/featureGeoJSON"
               }
            },
            "Schema featureCollectionGeoJSON links property":{
               "type":"array",
               "items":{
                  "$ref":"#/components/schemas/link"
               }
            },
            "Schema featureCollection GeoJSON timeStamp property":{
               "$ref":"$ref: '#/components/schemas/timeStamp'"
            },
            "Schema featureCollectionGeoJSON numberMatched property":{
               "$ref":"#/components/schemas/numberMatched"
            },
            "Schema featureCollectionGeoJSON numberReturned property":{
               "$ref":"$ref: '#/components/schemas/numberReturned'"
            }
         }
      },
      "featureGeoJSON":{
         "type":"object",
         "required":[
            "- geometry",
            "- properties",
            "- type"
         ],
         "properties":{
            "type":{
               "type":"string",
               "enum":"- feature"
            },
            "Schema featureGeoJSON geometry property":{
               "$ref":"$ref: '#/components/schemas/geometryGeoJSON'"
            },
            "Schema featureGeoJSON properties property":{
               "type":"object",
               "nullable":true
            },
            "Schema featureGeoJSON id properties":{
               "schema":{
                  "oneOf":[
                     "oneOf",
                     {
                        "type":"string"
                     },
                     {
                        "type":"integer"
                     }
                  ]
               }
            }
         }
      },
      "geometryGeoJSON":{
         "oneOf":[
            {
               "$ref":"#/components/schemas/pointGeoJSON"
            },
            {
               "$ref":"#/components/schemas/multipointGeoJSON"
            },
            {
               "$ref":"#/components/schemas/linestringGeoJSON"
            },
            {
               "$ref":"#/components/schemas/multilinestringGeoJSON"
            },
            {
               "$ref":"#/components/schemas/polygonGeoJSON"
            },
            {
               "$ref":"#/components/schemas/multipolygonGeoJSON"
            },
            {
               "$ref":"#/components/schemas/geometrycollectionGeoJSON"
            }
         ]
      },
      "geometrycollectionGeoJSON":{
         "type":"object",
         "required":[
            "- geometry",
            "- type"
         ],
            "properties":{
               "type":{
                  "enum":[
                     "- GeometryCollection"
                  ]
               }
            }
         },
         "landingPage":{
            "type":"object",
            "required":[
               "- links"
            ],
            "properties":{
               "Schema landingPage title property":{
                  "type":"string",
                  "example":"Buildings in Bonn"
               },
               "Schema landingPage description property":{
                  "type":"string",
                  "example":">-\n            Access to data about buildings in the
city of Bonn via a Web API\n            that conforms to the OGC API Features
specification."
               },
               "Schema landingPage links property":{
                  "type":"array",
                  "items":{
                     "$ref":"#/components/schemas/link"
                  }
               }
            }
         },
         "linestringGeoJSON":{
            "type":"object",
            "properties":{
               "type":{
                  "enum":"- LineString"
               },
               "Schema linestringGeoJSON coordinates property":{
                  "minItems":2,
                  "type":[
                     "array",
                     {
                        "type":"number"
                     }
                  ]
               }
            }
         },
         "link":{
            "type":"object",
            "required":[
               "- href"
            ],
            "properties":{
               "Schema link href property":{
                  "type":"object",
                  "example":"http://data.example.com/buildings/123"
               },
               "Schema link rel property":{
                  "type":"string",
                  "example":"alternate"
               },
               "type":{
                  "example":"application/geo+json"
               },
               "Schema link hreflang property":{
                  "type":"string",
                  "example":"en"
               },
               "Schema link title property":{
                  "type":"string",
                  "example":"Trierer Strasse 70, 53115 Bonn"
               },
               "Schema link length property":{
                  "type":"integer"
               }
            }
         },
         "multilinestringGeoJSON":{
            "type":"object",
            "required":[
               "- type",
               "- coordinates"
            ],
            "properties":{
               "type":{
                  "enum":[
                     "- MultiLineString"
                  ]
               },
               "Schema multilinestringGeoJSON coordinates property":{
                  "type":"array",
                  "items":{
                     "minItems":2,
                     "type":"array",
                     "items":{
                        "minItems":2,
                        "type":"array",
                        "items":{
                           "type":"number"
                        }
                     }
                  }
               }
            }
         },
         "multipointGeoJSON":{
            "type":"object",
            "properties":{
               "Schema multilinestringGeoJSON coordinates property":{
                  "type":"array",
                  "items":{
                     "minItems":2,
                     "type":"array",
                     "items":{
                        "minItems":2,
                        "type":"array",
                        "items":{
                           "type":"number"
                        }
                     }
                  }
               },
               "type":{
                  "enum":"- multipoint"
               }
            }
         },
         "multipolygonGeoJSON":{
            "type":"object",
            "required":[
               "- coordinates",
               "- type"
            ],
            "properties":{
               "type":{
                  "enum":"- MultiPolygon"
               },
               "Schema multipolygonGeoJSON coordinates property":{
                  "type":"array",
                  "items":{
                     "type":"array",
                     "items":{
                        "type":"array",
                        "minItems":4,
                        "items":{
                           "minItems":2,
                           "type":"array",
                           "items":{
                              "type":"number"
                           }
                        }
                     }
                  }
               }
            }
         },
         "numberMatched":{
            "minimum":0,
            "type":"integer",
            "description":"|-\n        The number of features of the feature type that
match the selection\n        parameters like `bbox`.",
            "example":127
         },
         "numberReturned":{
            "minimum":0,
            "type":"integer",
            "example":10
         },
         "pointGeoJSON":{
            "type":"object",
            "required":[
               "- coordinates",
               "- type"
            ],
            "properties":{
               "type":{
                  "enum":"- Point"
               },
               "Schema pointGeoJSON coordinates property":{
                  "minItems":2,
                  "type":"array",
                  "items":{
                     "type":"number"
                  }
               }
            }
         },
         "timeStamp":{
            "type":"string",
            "description":">-\n        This property indicates the time and date when
the response was\n        generated."
         }
      },
      "responses":{
         "LandingPage":[
            {
               "description":"The landing page provides links to the API definition\n
(link relations `service-desc` and `service-doc`),\n        the Conformance
declaration (path `/conformance`,\n        link relation `conformance`), and the
Feature\n        Collections (path `/collections`, link relation\n        `data`).",
               "content":[
                  {
                     "application/json":{
                        "schema":{
                           "$ref":"#/components/schemas/landingPage"
                        }
                     }
                  },
                  {
                     "name":"text/html"
                  }
               ]
            }
         ],
         "Collection":[
            {
               "description":"Information about the feature collection with id
`collectionId`.\n\n\n        The response contains a link to the items in the
collection\n\n        (path `/collections/{collectionId}/items`, link relation
`items`)\n\n        as well as key information about the collection. This
information",
               "content":{
                  "application/json":{
                     "schema":{
                        "$ref":"#/components/schemas/collection"
                     }
                  }
               }
            }
         ],
         "Queryables":[
            {
               "description":"Information about the feature collection queryable
properties",
               "content":{
                  "schema":{
                     "$ref":"#/components/schemas/queryables"
                  }
               }
            }
         ],
         "FilterCapabilities":[
            {
               "description":"A document listing the server filtering capabilities",
               "content":[
                  {
                     "name":"text/html"
                  },
                  {
                     "schema":{

                     }
                  }
```

```
            ]
         }
      ],
      "ConformanceDeclaration":[
         {
            "description":"The URIs of all conformance classes supported by the
server.\n\n        To support \"generic\" clients that want to access multiple\n
OGC API Features implementations - and not \"just\" a specific\n        API / server,
the server declares the conformance\n        classes it implements and conforms to.",
            "content":[
               {
                  "name":"text/html"
               },
               {
                  "application/json":{
                     "schema":{
                        "$ref":"#/components/schemas/confClasses"
                     }
                  }
               }
            ]
         }
      ],
      "Feature":[
         {
            "description":"fetch the feature with id `featureId` in the feature
collection\n        with id `collectionId`",
            "content":{
               "application/geo+json":{
                  "schema":{
                     "$ref":"#/components/schemas/featureGeoJSON"
                  }
               }
            }
         }
      ],
      "Features":[
         {
            "description":">-\n        The response is a document consisting of
features in the collection.\n\n        The features included in the response are
determined by the server\n\n        based on the query parameters of the request. To
support access to\n\n        larger collections without overloading the client, the
API supports\n\n        paged access with links to the next page, if more features are
selected\n\n        that the page size.\n\n\n        The `bbox` and `datetime`
parameter can be used to select only a\n\n        subset of the features in the
collection (the features that are in the\n\n        bounding box or time interval).
The `bbox` parameter matches all\n        features\n\n        in the collection that
are not associated with a location, too. The\n\n        `datetime` parameter matches
all features in the collection that are\n\n        not associated with a time stamp or
interval, too.\n\n\n        The `limit` parameter may be used to control the subset of
the\n\n        selected features that should be returned in the response, the page\n
```

```
size.\n\n        Each page may include information about the number of selected
and\n\n        returned features (`numberMatched` and `numberReturned`) as well as\n\n
links to support paging (link relation `next`).",
               "content":{
                  "schema":{
                     "$ref":"#/components/schemas/featureGeoJSON"
                  }
               }
            }
         ],
         "Collections":[
            {
               "description":"The feature collections shared by this API.\n\n\n
The dataset is organized as one or more feature collections. This\n
resource\n\n        provides information about and access to the collections.\n\n\n
The response contains the list of collections. For each collection, a\n
link\n\n        to the items in the collection (path\n
`/collections/{collectionId}/items`,\n\n        link relation `items`) as well as key
information about the collection.",
               "content":[
                  {
                     "name":"text/html"
                  },
                  {
                     "application/json":{
                        "schema":[
                           {
                              "$ref":"#/components/responses/Collections"
                           },
                           {
                              "$ref":"#/components/schemas/collections"
                           }
                        ]
                     }
                  }
               ]
            }
         ],
         "ServerError":[
            {
               "description":"A server error occurred.",
               "content":[
                  {
                     "name":"text/html"
                  },
                  {
                     "application/json":{
                        "schema":{
                           "$ref":"#/components/schemas/exception'"
                        }
                     }
```

```
                }
             ]
          }
       ],
       "InvalidParameter":[
          {
             "description":"A query parameter has an invalid value.",
             "content":{
                "schema":{
                   "$ref":"#/components/schemas/exception'"
                }
             }
          }
       ]
    },
    "parameters":{
       "bbox":{
          "name":"bbox",
          "in":"query",
          "description":">-\n        Only features that have a geometry that
intersects the bounding box are\n        selected.\n\n        The bounding box is
provided as four or six numbers, depending on\n        whether the\n\n
coordinate reference system includes a vertical axis (height or depth):\n\n\n        *
Lower left corner, coordinate axis 1\n\n        * Lower left corner, coordinate axis
2\n\n        * Minimum value, coordinate axis 3 (optional)\n\n        * Upper right
corner, coordinate axis 1\n\n        * Upper right corner, coordinate axis 2\n\n
* Maximum value, coordinate axis 3 (optional)\n\n\n        The coordinate reference
system of the values is WGS 84\n        longitude/latitude\n\n
(http://www.opengis.net/def/crs/OGC/1.3/CRS84) unless a different\n
coordinate\n\n        reference system is specified in the parameter `bbox-crs`.\n\n\n
For WGS 84 longitude/latitude the values are in most cases the sequence\n
of\n\n        minimum longitude, minimum latitude, maximum longitude and maximum\n
latitude.\n\n        However, in cases where the box spans the antimeridian the first
value\n\n        (west-most box edge) is larger than the third value (east-most box\n
edge).\n\n\n        If the vertical axis is included, the third and the sixth number
are\n\n        the bottom and the top of the 3-dimensional bounding box.\n\n\n
If a feature has multiple spatial geometry properties, it is the\n        decision of
the\n\n        server whether only a single spatial geometry property is used to\n
determine\n\n        the extent or all relevant geometries.",
          "required":false,
          "style":"form",
          "explode":false,
          "schema":{
             "maxItems":6,
             "minItems":4,
             "type":"array",
             "items":{
                "type":"number"
             }
          }
       },
         "CollectionId":{
            "name":"CollectionId",
            "in":"path",
            "description":"local identifier of a collection",
            "required":true,
            "schema":{
               "type":"string",
               "enum":[
                  "- 'syria_vtp:building_s'\n          - 'syria_vtp:built_up_area_s'\n
- 'syria_vtp:cemetery_s'\n          - 'syria_vtp:crop_land_s'\n          -
'syria_vtp:dam_s'\n          - 'syria_vtp:electric_power_station_s'\n          -
'syria_vtp:facility_s'\n          - 'syria_vtp:grassland_s'\n          -
'syria_vtp:military_installation_s'\n          - 'syria_vtp:power_substation_s'\n
- 'syria_vtp:river_c'\n          - 'syria_vtp:river_s'\n          -
'syria_vtp:road_c'\n          - 'syria_vtp:settlement_s'\n          -
'syria_vtp:tower_s'\n          - 'vtp:AeronauticPnt'\n          -
'vtp:AgriculturePnt'\n          - 'vtp:AgricultureSrf'\n          - 'vtp:CulturePnt'\n
- 'vtp:CultureSrf'\n          - 'vtp:FacilityPnt'\n          - 'vtp:HydrographyCrv'\n
- 'vtp:HydrographyPnt'\n          - 'vtp:HydrographySrf'\n          -
'vtp:MilitarySrf'\n          - 'vtp:SettlementSrf'\n          - 'vtp:StoragePnt'\n
- 'vtp:StructurePnt'\n          - 'vtp:TransportationGroundCrv'\n          -
'vtp:UtilityInfrastructureCrv'\n          - 'vtp:UtilityInfrastructurePnt'\n
- 'vtp:VegetationSrf'\n          - 'iraq_vtp:aircraft_hangar_s'\n          -
'iraq_vtp:amusement_park_s'\n          - 'iraq_vtp:annotated_location_s'\n          -
'iraq_vtp:apron_s'\n          - 'iraq_vtp:archeological_site_s'\n          -
'iraq_vtp:barn_s'\n          - 'iraq_vtp:bridge_c'\n          - 'iraq_vtp:bridge_s'\n
- 'iraq_vtp:brush_s'\n          - 'iraq_vtp:building_p'\n          -
'iraq_vtp:building_s'\n          - 'iraq_vtp:built_up_area_p'\n          -
'iraq_vtp:built_up_area_s'\n          - 'iraq_vtp:canal_c'\n          -
'iraq_vtp:cart_track_c'\n          - 'iraq_vtp:castle_s'\n          -
'iraq_vtp:cemetery_s'\n          - 'iraq_vtp:crop_land_s'\n          -
'iraq_vtp:crossing_p'\n          - 'iraq_vtp:culvert_c'\n          -
'iraq_vtp:cut_c'\n          - 'iraq_vtp:dam_c'\n          - 'iraq_vtp:dam_s'\n
- 'iraq_vtp:disposal_site_s'\n          - 'iraq_vtp:ditch_c'\n          -
'iraq_vtp:embankment_c'\n          - 'iraq_vtp:extraction_mine_s'\n          -
'iraq_vtp:facility_s'\n          - 'iraq_vtp:fence_c'\n          -
'iraq_vtp:firing_range_s'\n          - 'iraq_vtp:ford_c'\n          -
'iraq_vtp:forest_s'\n          - 'iraq_vtp:fountain_s'\n          -
'iraq_vtp:gate_c'\n          - 'iraq_vtp:gate_p'\n          -
'iraq_vtp:grain_storage_structure_s'\n          - 'iraq_vtp:grassland_s'\n          -
'iraq_vtp:greenhouse_s'\n          - 'iraq_vtp:helipad_p'\n          -
'iraq_vtp:helipad_s'\n          - 'iraq_vtp:hut_s'\n          -
'iraq_vtp:hydrocarbons_field_s'\n          - 'iraq_vtp:inland_waterbody_s'\n
- 'iraq_vtp:interest_site_p'\n          - 'iraq_vtp:interest_site_s'\n          -
'iraq_vtp:island_s'\n          - 'iraq_vtp:land_aerodrome_p'\n          -
'iraq_vtp:land_aerodrome_s'\n          - 'iraq_vtp:land_water_boundary_c'\n          -
'iraq_vtp:lookout_s'\n          - 'iraq_vtp:marsh_s'\n          -
'iraq_vtp:memorial_monument_s'\n          - 'iraq_vtp:military_installation_s'\n
- 'iraq_vtp:motor_vehicle_station_p'\n          - 'iraq_vtp:motor_vehicle_station_s'\n
- 'iraq_vtp:orchard_s'\n          - 'iraq_vtp:park_s'\n          -
'iraq_vtp:power_substation_s'\n          - 'iraq_vtp:racetrack_c'\n          -
'iraq_vtp:racetrack_s'\n          - 'iraq_vtp:railway_c'\n          -
'iraq_vtp:railway_sidetrack_c'\n          - 'iraq_vtp:river_c'\n          -
'iraq_vtp:river_s'\n          - 'iraq_vtp:road_c'\n          - 'iraq_vtp:road_s'\n
- 'iraq_vtp:roadside_rest_area_s'\n          - 'iraq_vtp:ruins_p'\n          -
'iraq_vtp:ruins_s'\n          - 'iraq_vtp:settlement_s'\n          -
'iraq_vtp:sewage_treatment_plant_s'\n          - 'iraq_vtp:shed_s'\n          -
'iraq_vtp:shopping_complex_s'\n          - 'iraq_vtp:sports_ground_s'\n          -
'iraq_vtp:stadium_s'\n          - 'iraq_vtp:stair_c'\n          -
'iraq_vtp:steep_terrain_face_c'\n          - 'iraq_vtp:storage_tank_s'\n          -
'iraq_vtp:swamp_s'\n          - 'iraq_vtp:swimming_pool_s'\n          -
'iraq_vtp:taxiway_c'\n          - 'iraq_vtp:tower_s'\n          -
'iraq_vtp:traffic_light_p'\n          - 'iraq_vtp:trail_c'\n          -
'iraq_vtp:transportation_block_p'\n          - 'iraq_vtp:transportation_station_s'\n
- 'iraq_vtp:tunnel_c'\n          - 'iraq_vtp:vehicle_barrier_c'\n          -
'iraq_vtp:vehicle_lot_s'\n          - 'iraq_vtp:wall_c'\n          -
'iraq_vtp:water_tower_s'\n          - 'iraq_vtp:water_well_s'\n          -
'iraq_vtp:waterwork_s'\n          - 'iraq_vtp:zoo_s'\n          - 'ne:countries50m'\n
- 'ne:popplaces50m'\n          - 'ne:urban50m'"
               ]
            }
         },
         "dateTime":{
            "name":"datetime",
            "in":"query",
            "description":"Either a date-time or an interval, open or closed. Date and
time\n        expressions\n\n        adhere to RFC 3339. Open intervals are expressed
using double-dots.",
            "required":false,
            "style":"form",
            "explode":false,
            "schema":{
               "type":"string"
            }
         },
         "featureId":{
            "name":"featureId",
            "in":"path",
            "description":"local identifier of a feature",
            "required":true,
            "schema":{

            }
         },
         "limit":{
            "name":"limit",
            "in":"query",
            "description":"The optional limit parameter limits the number of items
that are\n        presented in the response document.\n\n\n        Only items are
counted that are on the first level of the collection in\n        the response
document.\n\n        Nested objects contained within the explicitly requested items
shall not\n        be counted.\n\n\n        Minimum = 1. Maximum = 10000. Default =
10.",
            "required":false,
            "style":false,
            "explode":false,
            "schema":{
               "maximum":1000000,
               "minimum":1,
               "type":"integer",
               "default":1000000
            }
         },
         "filter":{
            "name":"filter",
            "in":"query",
            "description":">-\n        Defines a filter that will be applied on items,
only items matching the\n        filter will be returned",
            "schema":{
               "type":"string"
            }
         },
         "filter-lang":{
            "name":"filter-lang",
            "in":"query",
            "description":"Filter encoding used in the filter parameter",
            "schema":{
               "type":"string",
               "enum":[
                  "cql-text"
               ],
               "default":"cql-text"
            }
         }
      }
   }
}
```

Appendix C: OpenAPI Extension task

This section reports on the work completed as part of an unfunded OpenAPI extension task after Testbed-16 work was completed. The approach taken was to start the development of ShapeChange as a new project omitting all of the work completed in Testbed-16, essentially the Testbed-16 work was completed for a second time. The lessons learned from Testbed-16 were applied in this extension work to develop a new implementation with a fundamentally new approach. Additionally, the UML model was changed and troublesome data types and classes from the existing metamodel were changed to accommodate ShapeChange requirements, specifically, the SCXML representation of the UML model. Additionally, the model was simplified where possible.

C.1. Recap of issues to be addressed

There were several shortcomings of the Testbed-16 approach to producing a conformant JSON-encoded definition of an OpenAPI interface, briefly, these were as follows:

- The use of association classes is a valid approach to describing certain relationships in OpenAPI and although SCXML can *see* these classes, the relationship they refer to is not clear.
- The use of enumerations was unclear, particularly in the *responses* classes. This is because the OpenAPI specification defines this as a *variable* when in fact it is fact a controlled vocabulary consisting of approved HTTP status codes.
- The ShapeChange implementation was focused on *producing a compliant result*, rather than specifically implementing tests against the metamodel.

C.2. Addressing the shortcomings

As mentioned previously, the work done in Testbed-16 was discarded and implementation was started from scratch. Changes to the UML model and metamodel were also made, these are documented in this section. This work was all completed using internal innovation funding and does not form part of Testbed-16.

C.2.1. UML model Changes

There are two main changes that were made to the UML model and metamodel in this implementation, these were:

- All relationships described with association classes has been removed, these relationships are now described with *qualifications*.

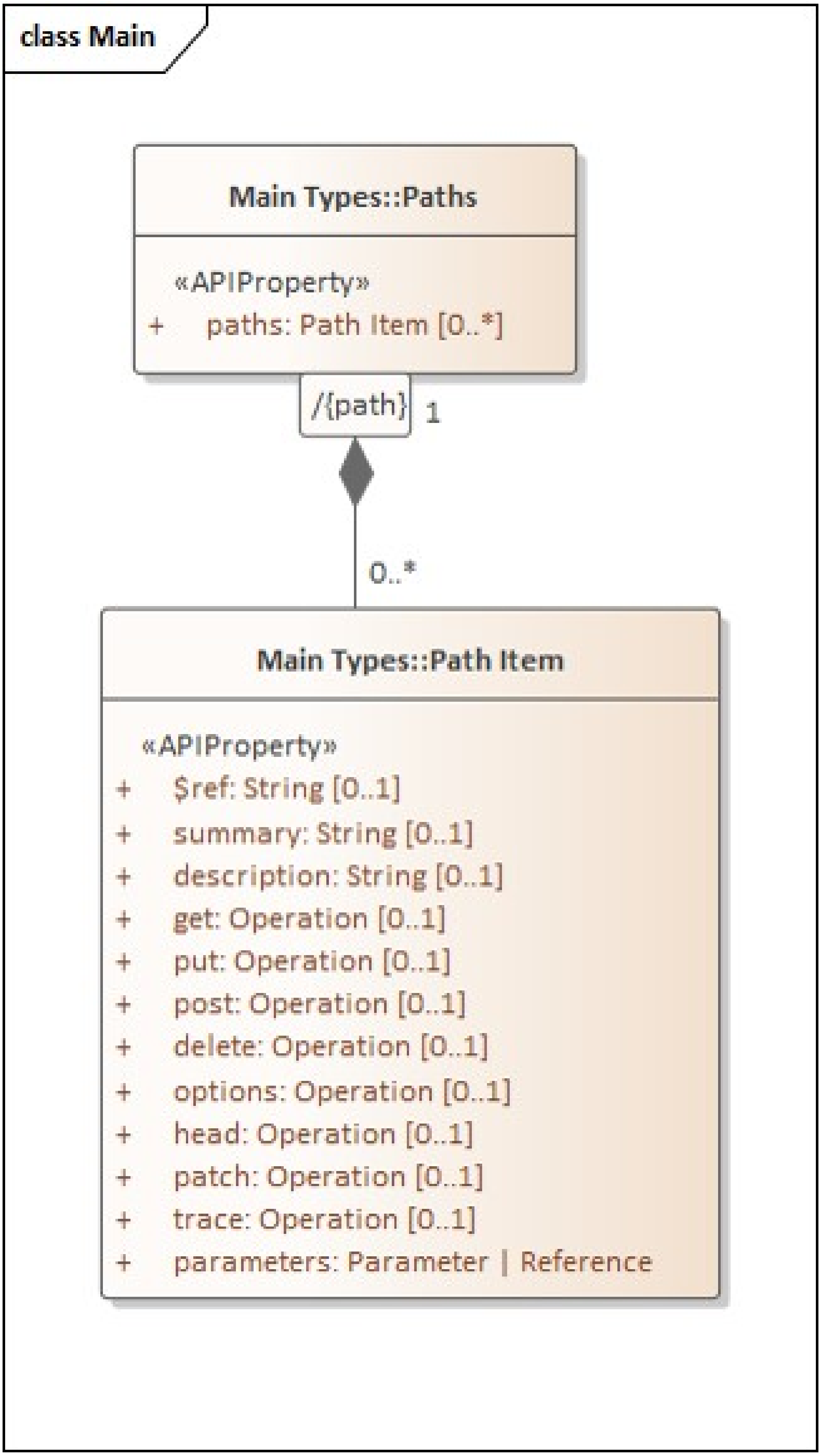

class Main
Main Types::Paths
«APIProperty»
+ paths: Path Item [0..*]
/{path}
1
0..*
Main Types::Path Item
«APIProperty»
+ $ref: String [0..1]
+ summary: String [0..1]
+ description: String [0..1]
+ get: Operation [0..1]
+ put: Operation [0..1]
+ post: Operation [0..1]
+ delete: Operation [0..1]
+ options: Operation [0..1]
+ head: Operation [0..1]
+ patch: Operation [0..1]
+ trace: Operation [0..1]
+ parameters: Parameter | Reference

Figure 26. Relationships as qualifications

- The responses class now contains all of the HTTP Response codes as variables with multiplicity of 0 or 1. The specification describes this relationship as a value, value combination (rather than a key, value pair). This is a similar approach to the *Operation* class, which contains all of the possible http verbs with a multiplicity of 0..1.

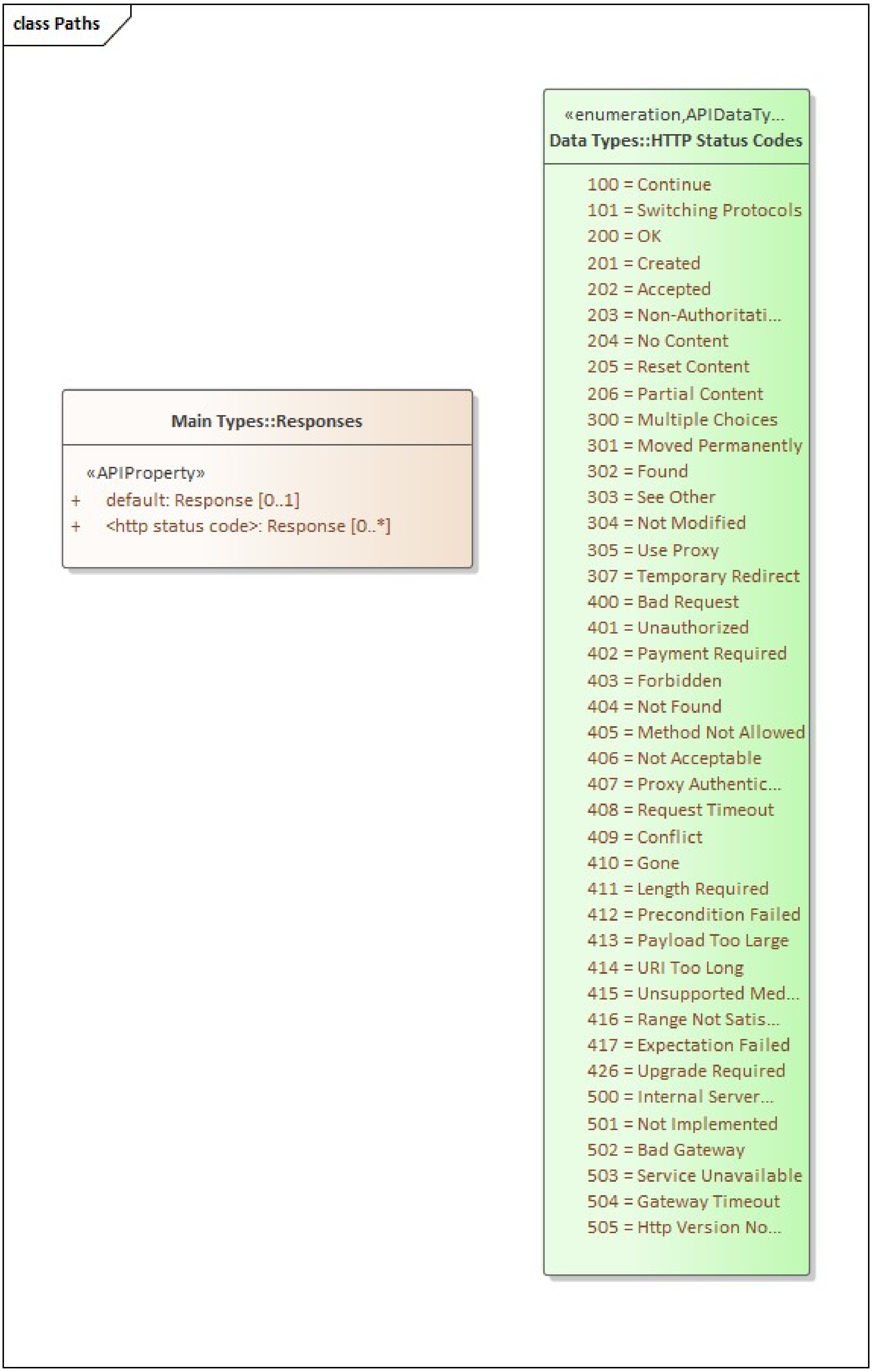


Figure 27. Responses

- A general tidy up of the UML metamodel to reflect the changes described and correct any errors. It was noticed in the second round of implementation that the metamodel contained a few relationship errors, these were corrected for completeness as in reality, ShapeChange only uses the *specialization definition* at this time and does not use the attributes in the metamodel.

C.2.2. ShapeChange implementation

The Testbed-16 ShapeChange implementation used a combination of generalizations and class names to produce the output, this was changed in this implementation to only use the generalized relationships for consistency. Therefore, the rules were applied at the metamodel level only and any *model* based rules were removed. This represented a larger development overhead, but produces better results. Although this development work produces the required result, it does not represent a full implementation of the OpenAPI specification into ShapeChange, however, the approach taken appears to provide a solid pathway for doing so.

C.2.3. Testing

Testing in the initial Testbed-16 work was restricted to a single implementation of OGC API - Features - Part 1: Core. This resulted in overfitting the implementation to a single specification and therefore solved a single case but led to developmental dead-ends. Therefore, a set of tests were devised for the extension implementation, these were:

- A very simple OpenAPI minimal example provided by the OpenAPI team on the website.
- A Helyx-specified OpenAPI interface designed for a different project and not OGC compliant.
- A minimal implementation of OGC API - Features - Part 1: Core.

The objective was to create a generic model and solution that fits all three of these use cases. The three use cases contained a usage of references, internally defined schemas, parameters, multiple paths and components to provide confidence that the approach taken solves the generic use case.

C.2.4. Results

By making the changes outlined, it was possible to create a generic solution for the three use cases.

C.2.4.1. Example minimal OpenAPI

The initial test was the sample API taken from the OpenAPI website as a minimal implementation. The UML model for this is simple and linear and contains only a single response. Note the changes from the Testbed-16 work in the Responses class which has a variable named *200*, this reflects the fact that the Responses metaclass contains a controlled vocabulary of http responses. Additionally, the *Paths* class only has one *path* described by a UML qualification.

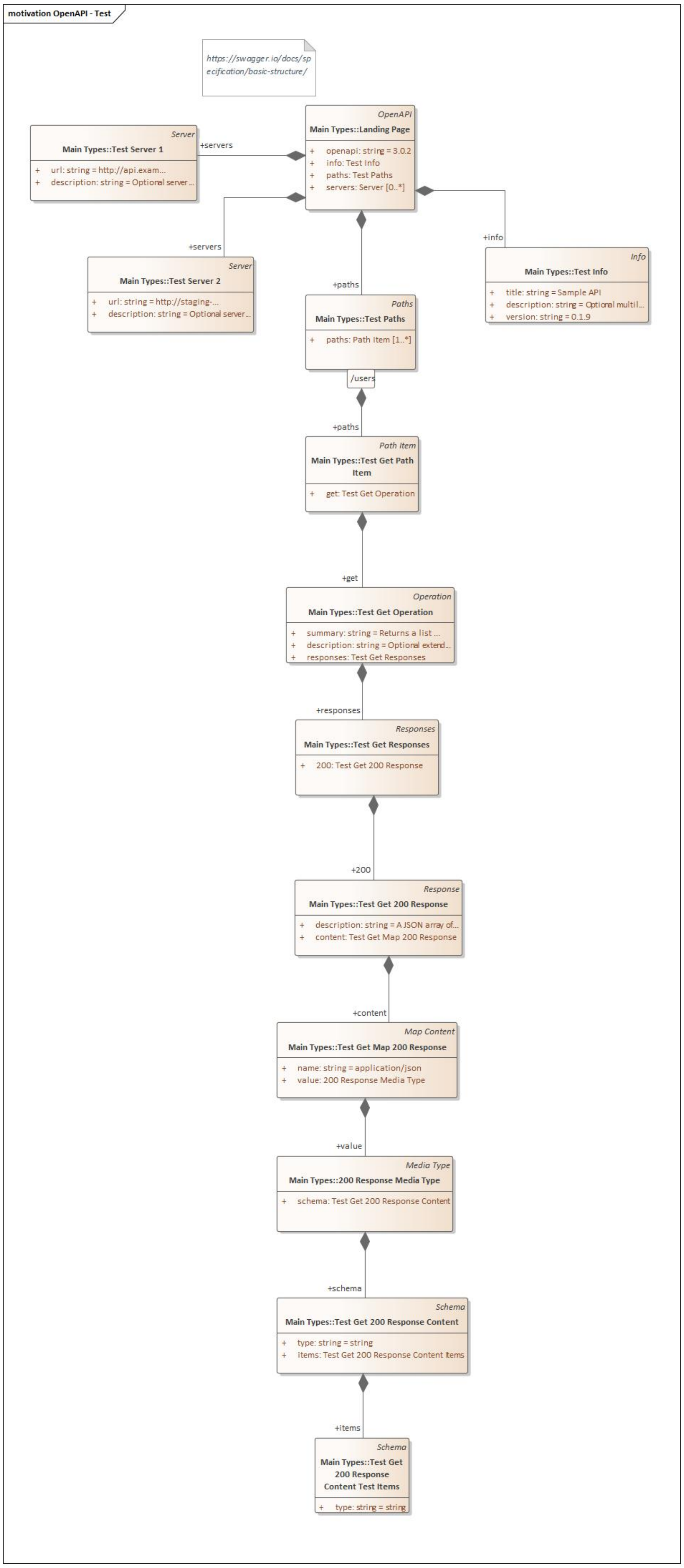

motivation OpenAPI - Test
https://swagger.io/docs/specification/basic-structure/
OpenAPI
Main Types::Landing Page
+ openapi: string = 3.0.2
+ info: Test Info
+ paths: Test Paths
+ servers: Server [0..*]
Server
Main Types::Test Server 1
+ url: string = http://api.exam...
+ description: string = Optional server...
+servers
+servers
Server
Main Types::Test Server 2
+ url: string = http://staging-...
+ description: string = Optional server...
+info
Info
Main Types::Test Info
+ title: string = Sample API
+ description: string = Optional multil...
+ version: string = 0.1.9
+paths
Paths
Main Types::Test Paths
+ paths: Path Item [1..*]
/users
+paths
Path Item
Main Types::Test Get Path Item
+ get: Test Get Operation
+get
Operation
Main Types::Test Get Operation
+ summary: string = Returns a list ...
+ description: string = Optional extend...
+ responses: Test Get Responses
+responses
Responses
Main Types::Test Get Responses
+ 200: Test Get 200 Response
+200
Response
Main Types::Test Get 200 Response
+ description: string = A JSON array of...
+ content: Test Get Map 200 Response
+content
Map Content
Main Types::Test Get Map 200 Response
+ name: string = application/json
+ value: 200 Response Media Type
+value
Media Type
Main Types::200 Response Media Type
+ schema: Test Get 200 Response Content
+schema
Schema
Main Types::Test Get 200 Response Content
+ type: string = string
+ items: Test Get 200 Response Content Items
+items
Schema
Main Types::Test Get 200 Response Content Test Items
+ type: string = string

Figure 28. OpenAPI Test taken from https://swagger.io/docs/specification/basic-structure/

```
{
    "openapi": "3.0.2",
    "info": {
        "title": "Sample API",
        "description": "Optional multiline or single-line description in
[CommonMark](http://commonmark.org/help/) or HTML.",
        "version": "0.1.9"
    },
    "servers": [
        {
            "url": "http://api.example.com/v1",
            "description": "Optional server description, e.g. Main (production)
server"
        },
        {
            "url": "http://staging-api.example.com",
            "description": "Optional server description, e.g. Internal staging server
for testing"
        }
    ],
    "paths": [{"/users": {"get": {
        "summary": "Returns a list of users.",
        "description": "Optional extended description in CommonMark or HTML.",
        "responses": {
            "200": {
                "description": "A JSON array of user names",
                "content": {"application/json": {"schema": {
                    "type": "string",
                    "items": {"type": "string"}
                }}}
            },
        }
    }}}]
}
```

C.2.4.2. Example microservice OpenAPI

This example is taken from a microservice definition that is more complex than the previous example, but not OGC compliant. It contains references as well as definitions for schemas and multiple paths and responses defined in the described way.

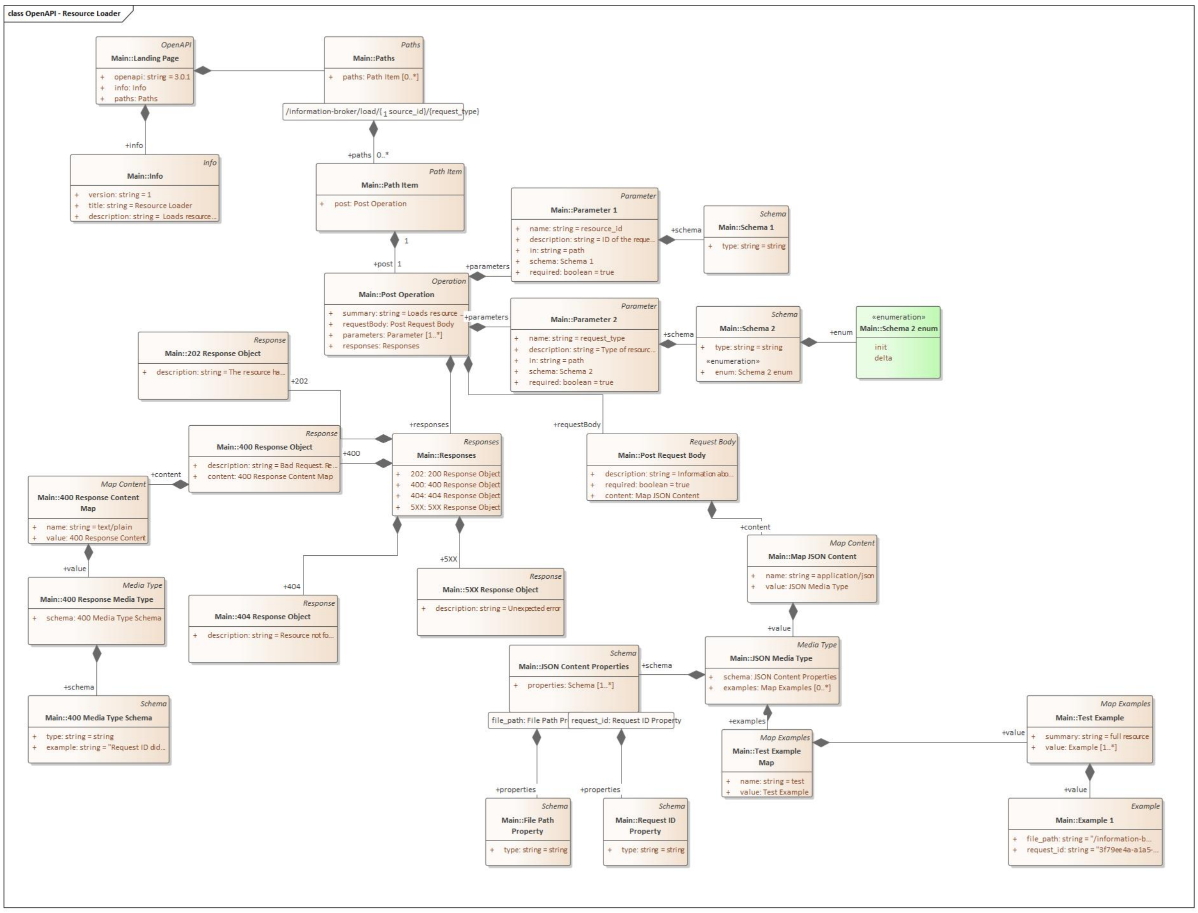


Figure 29. OpenAPI definition for a Resource Loader

```
{
      "openapi": "3.0.1",
      "info": {
            "version": "1",
            "title": "Resource Loader",
            "description": "Loads resources from the local NFS to the appropriate
local service, informs the local catalogue of the change and notifies the Resource
Requester that a request has been fulfilled."
      },
      "paths": {"/information-broker/load/{resource_id}/{request_type}": {"post": {
            "summary": "Loads resource from local NFS to appropriate local service.",
            "requestBody": {
                  "description": "Information about the request and the resource
location on the local NFS",
                  "required": true,
                  "content": {"application/json": {
                        "schema": {"properties": {
                              "file_path": {"type": "string"},
                              "request_id": {"type": "string"}
                        }},
                        "examples": {"test": {"summary": "full resource"}}
                  }}
            },
            "parameters": [
                  {
                        "name": "resource_id",
                        "description": "ID of the requested resource",
                        "in": "path",
                        "required": true,
                        "schema": {"type": "string"}
                  },
                  {
                        "name": "request_type",
                        "description": "Type of resource requested",
                        "in": "path",
                        "required": true,
                        "schema": {
                              "type": "string",
                              "enum": [{
                                  "init": "",
                                  "delta": ""
                              }]
                        }
                  }
            ],
            "responses": {
                  "202": {"description": "The resource has been located and will be
loaded into the appropriate service."},
                  "400": {
                        "description": "Bad Request. Reason given in response body",
                        "content": {"text/plain": {"schema": {
                              "type": "string",
                              "example": "Request ID did not match any outstanding
request processes."
                        }}}
                  },
                  "404": {"description": "Resource not found at given file path"},
                  "5XX": {"description": "Unexpected error"}
            }
      }}}
}
```

C.2.4.3. Example OGC API - Features - Part 1: Core

OGC API - Features - Part 1: Core was the test case for the Testbed-16 work, however, the definition and corresponding model was very complex. Therefore, a simpler implementation of the standard was tested in the post-Testbed-16 work. The definition is heavily reliant on references to external objects such as conformance classes. This implementation is potentially closer to the *building blocks* idea described in the recommendations of the main part of this report.

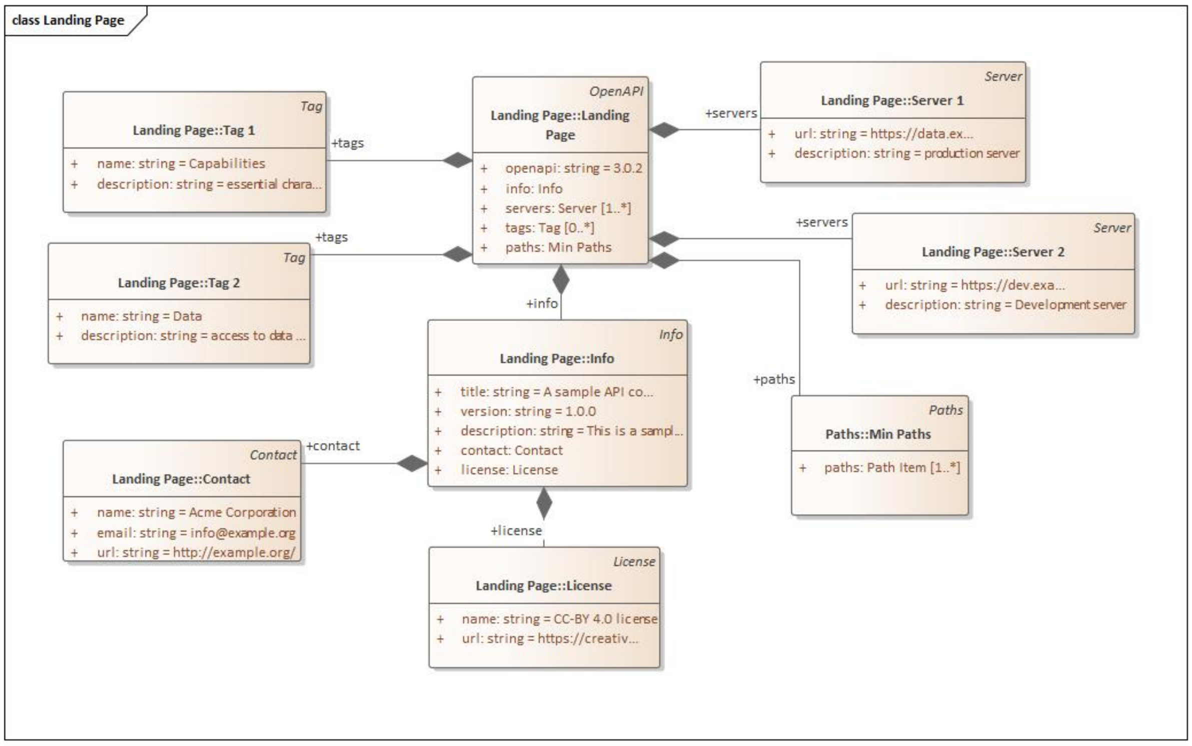

class Landing Page
Tag
Landing Page::Tag 1
+ name: string = Capabilities
+ description: string = essential chara...
+tags
OpenAPI
Landing Page::Landing Page
+ openapi: string = 3.0.2
+ info: Info
+ servers: Server [1..*]
+ tags: Tag [0..*]
+ paths: Min Paths
+servers
Server
Landing Page::Server 1
+ url: string = https://data.ex...
+ description: string = production server
+tags
Tag
Landing Page::Tag 2
+ name: string = Data
+ description: string = access to data ...
+servers
Server
Landing Page::Server 2
+ url: string = https://dev.exa...
+ description: string = Development server
+info
Info
Landing Page::Info
+ title: string = A sample API co...
+ version: string = 1.0.0
+ description: string = This is a sampl...
+ contact: Contact
+ license: License
+paths
Paths
Paths::Min Paths
+ paths: Path Item [1..*]
+contact
Contact
Landing Page::Contact
+ name: string = Acme Corporation
+ email: string = info@example.org
+ url: string = http://example.org/
+license
License
Landing Page::License
+ name: string = CC-BY 4.0 license
+ url: string = https://creativ...

Figure 30. Minimal OGC API-Feature Landing Page

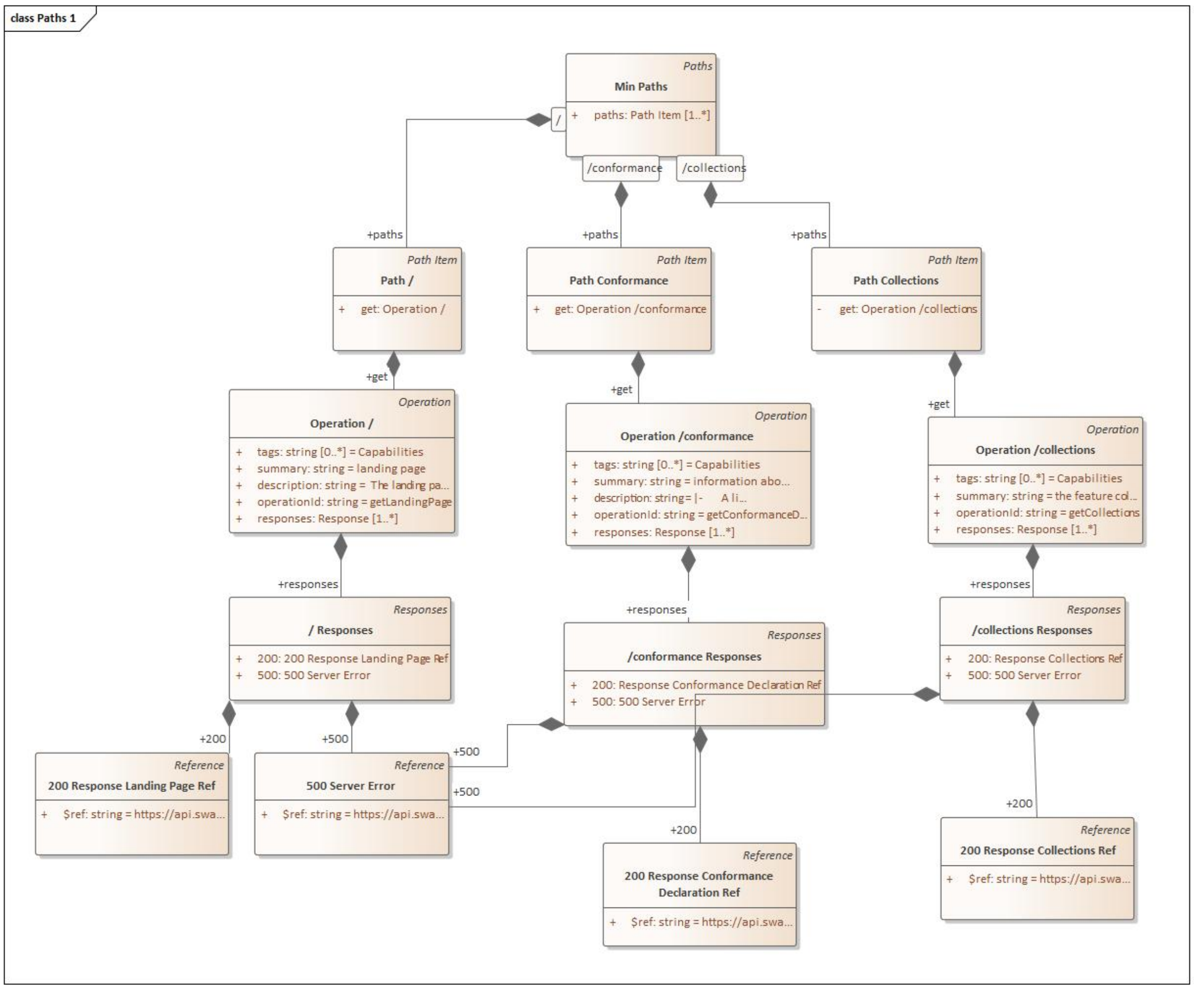

class Paths 1
Paths
Min Paths
+ paths: Path Item [1..*]
/
/conformance
/collections
+paths
+paths
+paths
Path Item
Path /
+ get: Operation /
Path Item
Path Conformance
+ get: Operation /conformance
Path Item
Path Collections
- get: Operation /collections
+get
+get
+get
Operation
Operation /
+ tags: string [0..*] = Capabilities
+ summary: string = landing page
+ description: string = The landing pa...
+ operationId: string = getLandingPage
+ responses: Response [1..*]
Operation
Operation /conformance
+ tags: string [0..*] = Capabilities
+ summary: string = information abo...
+ description: string = |- A li...
+ operationId: string = getConformanceD...
+ responses: Response [1..*]
Operation
Operation /collections
+ tags: string [0..*] = Capabilities
+ summary: string = the feature col...
+ operationId: string = getCollections
+ responses: Response [1..*]
+responses
+responses
+responses
Responses
/ Responses
+ 200: 200 Response Landing Page Ref
+ 500: 500 Server Error
Responses
/conformance Responses
+ 200: Response Conformance Declaration Ref
+ 500: 500 Server Error
Responses
/collections Responses
+ 200: Response Collections Ref
+ 500: 500 Server Error
+200
+500
+500
+500
+200
+200
Reference
200 Response Landing Page Ref
+ $ref: string = https://api.swa...
Reference
500 Server Error
+ $ref: string = https://api.swa...
Reference
200 Response Conformance Declaration Ref
+ $ref: string = https://api.swa...
Reference
200 Response Collections Ref
+ $ref: string = https://api.swa...

Figure 31. Minimal OGC API - Feature Paths

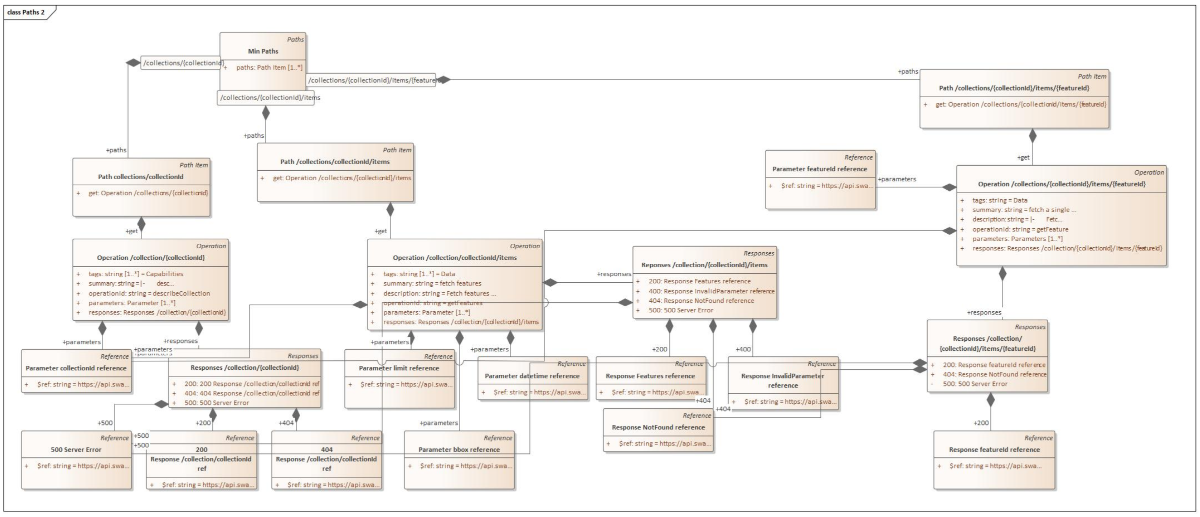


Figure 32. Minimal OGC API - Feature Paths continued

```
{
      "openapi": "3.0.2",
      "info": {
            "title": "A sample API conforming to the standard OGC API - Features -
Part 1: Core",
            "version": "1.0.0",
            "description": "This is a sample OpenAPI definition that conforms to the
conformance \n    classes \"Core\", \"GeoJSON\", \"HTML\" and \"OpenAPI 3.0\" of the\n
standard \"OGC API - Features - Part 1: Core\".\n    \n    This example is a generic
OGC API Features definition that uses path \n    parameters to describe all feature
collections and all features. \n    The generic OpenAPI definition does not provide
any details on the \n    collections or the feature content. This information is only
available \n    from accessing the feature collection resources.\n    \n    There is
\n    [another example](https://app.swaggerhub.com/apis/cportele/ogcapi-features-1-
example2/1.0.0) \n    that specifies each collection explicitly.",
            "contact": {
                  "name": "Acme Corporation",
                  "email": "info@example.org",
                  "url": "http://example.org/"
            },
            "license": {
                  "name": "CC-BY 4.0 license",
                  "url": "https://creativecommons.org/licenses/by/4.0/"
            }
      },
      "servers": [
            {
                  "url": "https://data.example.org/",
                  "description": "production server"
            },
            {
                  "url": "https://dev.example.org/",
                  "description": "Development server"
            }
      ],
      "tags": [
            {
                  "name": "Capabilities",
                  "description": "essential characteristics of this API"
            },
            {
                  "name": "Data",
                  "description": "access to data (features)"
            }
      ],
      "paths": {
            "/": {"get": {
                  "tags": ["Capabilities"],
                  "summary": "landing page",
                  "description": "The landing page provides links to the API
definition, the conformance\n        statements and to the feature collections in this
dataset.",
                  "operationId": "getLandingPage",
                  "responses": {
                        "200": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/LandingPage"},
                        "500": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ServerError"}
                  }
            }},
            "/conformance": {"get": {
                  "tags": ["Capabilities"],
                  "summary": "information about specifications that this API conforms
to",
                  "description": "|-\n        A list of all conformance classes
specified in a standard that the \n        server conforms to.",
                  "operationId": "getConformanceDeclaration",
                  "responses": {
                        "500": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ServerError"},
                        "200": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ConformanceDeclaration"}
                  }
            }},
            "/collections": {"get": {
                  "tags": ["Capabilities"],
                  "summary": "the feature collections in the dataset",
                  "operationId": "getCollections",
                  "responses": {
                        "500": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ServerError"},
                        "200": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/Collections"}
                  }
            }},
            "/collections/{collectionId}": {"get": {
                  "tags": ["Capabilities"],
                  "summary": "|-\n        describe the feature collection with id
`collectionId`",
                  "operationId": "describeCollection",
                  "parameters": [{"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/parameters/collectionId"}],
                  "responses": {
                        "500": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ServerError"},
                        "200": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/Collection"},
                        "404": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/NotFound"}
                  }
            }},
            "/collections/{collectionId}/items": {"get": {
                  "tags": ["Data"],
                  "summary": "fetch features",
                  "description": "Fetch features of the feature collection with id
`collectionId`.\n        \n        Every feature in a dataset belongs to a collection.
A dataset may\n        consist of multiple feature collections. A feature collection
is often a\n        collection of features of a similar type, based on a common
schema.\n\n        Use content negotiation to request HTML or GeoJSON.",
                  "operationId": "getFeatures",
                  "parameters": [
                        {"$ref": "https://api.swaggerhub.com/domains/cportele/ogcapi-
features-1/1.0.0#/components/parameters/collectionId"},
                        {"$ref": "https://api.swaggerhub.com/domains/cportele/ogcapi-
features-1/1.0.0#/components/parameters/limit"},
                        {"$ref": "https://api.swaggerhub.com/domains/cportele/ogcapi-
features-1/1.0.0#/components/parameters/bbox"},
                        {"$ref": "https://api.swaggerhub.com/domains/cportele/ogcapi-
features-1/1.0.0#/components/parameters/datetime"}
                  ],
                  "responses": {
                        "500": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ServerError"},
                        "200": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/Features"},
                        "400": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/InvalidParameter"},
                        "404": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/NotFound"}
                  }
            }},
            "/collections/{collectionId}/items/{featureId}": {"get": {
                  "tags": ["Data"],
                  "summary": "fetch a single feature",
                  "description": "|-\n        Fetch the feature with id `featureId` in
the feature collection\n        with id `collectionId`.\n\n        Use content
negotiation to request HTML or GeoJSON.",
                  "operationId": "getFeature",
                  "parameters": [
                        {"$ref": "https://api.swaggerhub.com/domains/cportele/ogcapi-
features-1/1.0.0#/components/parameters/collectionId"},
                        {"$ref": "https://api.swaggerhub.com/domains/cportele/ogcapi-
features-1/1.0.0#/components/parameters/featureId"}
                  ],
                  "responses": {
                        "500": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/ServerError"},
                        "404": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/NotFound"},
                        "200": {"$ref":
"https://api.swaggerhub.com/domains/cportele/ogcapi-features-
1/1.0.0#/components/responses/Feature"}
                  }
            }}
      }
}
```

C.2.5. Conclusion

The decision to reevaluate the UML modeling approach and reimplement the ShapeChange plugin to reflect the new design decisions resulted in a solid partial implementation of the approach into ShapeChange. It is believed that the approach taken in the extension work with the UML model and the ShapeChange implementation provides a pathway to a full, workable implementation for the generic case.

Appendix D: Revision History

Table 1. Revision History

Date	**Editor**	**Release**	**Primary clauses modified**	**Descriptions**
November 19, 2020	S. Meek	1.0	multiple	submitted version

Appendix E: Bibliography

[1] Echterhoff, J.: UML-to-GML Application Schema Pilot (UGAS-2020) Engineering Report. OGC 20-012,Open Geospatial Consortium, https://docs.ogc.org/per/ (2021).

[2] Echterhoff, J.: OGC Testbed-14: Application Schemas and JSON Technologies Engineering Report. OGC 18-091r2,Open Geospatial Consortium, https://docs.ogc.org/per/18-091r2.html (2018).