Testbed-12 ShapeChange Engineering Report

Table of contents

Publication Date: 2017-04-04

Approval Date: 2017-03-09

Posted Date: 2016-10-31

Reference number of this document: OGC 16-020

Reference URL for this document: http://www.opengis.net/doc/PER/t12-A087

Category: Public Engineering Report

Editor: Johannes Echterhoff

Title: Testbed-12 ShapeChange Engineering Report

OGC Engineering Report

COPYRIGHT

WARNING

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

LICENSE AGREEMENT

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

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

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

This license is effective until terminated. You may terminate it at any time by

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

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

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

Abstract

This document is a deliverable of the OGC Testbed 12. It describes the results of enhancing the tool *ShapeChange* in the following areas of processing an ISO 19109 conformant application schema:

1. Creating a schema profile - to support implementations that focus on a subset of the use cases in scope of the original application schema.
2. Deriving an ontology representation of the application schema (using RDF(S)/SKOS/OWL) - to support Semantic Web / Linked Data implementations.

Business Value

Application schemas are a key enabler of interoperable information exchange. They define the structure and semantics of geographic features for a specific domain, community, or application. Numerous application schemas exist, for example in the defense and intelligence as well as aviation domains.

Traditionally, XML Schemas have been derived from application schemas, based upon the encoding rules of the Geography Markup Language (GML). These schemas are used for exchanging XML encoded geographic information in an interoperable way.

This report advances location-based technologies in two ways:

- It describes the design of new functionality that allows a higher level of control and flexibility when creating an application schema profile.
- It defines rules for converting an application schema into an OWL ontology. The design is based upon the conversion rules defined by ISO 19150-2. A number of configuration options as well as additional conversion rules provide a higher level of control and flexibility when deriving an ontology compared to the conversion rules defined by ISO 19150-2.

Profiling:

Some application schemas are designed to support a range of applications and use cases. Often, software components are built to support a specific subset of these use cases. Likewise, products (like maps on different scales) may contain a specific subset of the information that is specified by the application schema. Profiling reduces the application schema to the relevant subset. Once a profile has been created, a number of implementation schemas can be derived, for example XML schemas, database schemas, and ontologies. These schemas serve

as blueprints and specifications that facilitate the implementation of geographic information systems. Note that the information to create a profile is stored in a machine readable way. A profile can thus automatically be created, which is a key benefit of model driven engineering. The new profiling functionality described in this report supports the creation of more fine-grained schema profiles. The existing profiling functionality supports the removal of irrelevant classes and properties from the application schema. With the new functionality, restrictions can be defined and applied on the remaining model elements. Furthermore, model constraints - for example defined by OCL expressions - can be added, updated, validated and deleted. With the new functionality designed in Testbed 12, a domain expert can create schema profiles that express the modeling intent in more detail than what has been possible before.

Conversion to OWL Ontology:

Converting an application schema into an ontology results in a key component that can be used by web applications. The ontology defines the concepts for encoding geographic information in machine-processible representation languages (RDF/OWL/SKOS). RDF data published on the web supports linking between different datasets. The ontology makes conceptual knowledge available for automated reasoning over RDF data. Combined, this can unlock new information.

The ER describes rules for deriving ontologies from ISO 19109 conformant application schemas. The encoding rules of the recently published ISO IS 19150-2 are taken into account and adapted as necessary.

The rules to convert an application schema to an OWL ontology have been implemented by ShapeChange. Application schemas can now automatically be converted to OWL ontologies. This facilitates the use of geographic information in linked data and semantic web applications.

The concepts defined by the ontology can also be used to provide meaning to the data contained in a JSON-LD document. JSON-LD adds linking to JSON, which is a popular data format for web developers.

Why this ER is valuable for the Working Group(s) and OGC in general

The ER is important for the OGC Architecture DWG and the OGC in general because it describes enhancements for working with application schemas in two areas: Profiling and UML to RDF/OWL/SKOS. These enhancements extend the

existing rules for modelling and encoding an application schema:

- Value of improvements in profiling. New features in the area of schema profiling support requirements of domains with large-scale application schemas. Profiling automates the task of creating a subset of an application schema to tailor it to the needs of a particular implementation. The result is improved mission-specific profiles created using a configurable, repeatable automated procedure.
- Value of UML-to-OWL transformation. New rules implemented in ShapeChange provide automated capability for deriving ontologies and controlled vocabularies in W3C encodings from ISO 19109 conformant application schemas. The result is support for the use of semantics to enhance geospatial information in Linked Data and Semantic Web applications.

Keywords

UML, RDF, OWL, SKOS, Profile, Profiling, ShapeChange, UGAS, Ontology, Controlled Vocabulary

Proposed OGC Working Group for Review and Approval

- Primary: Architecture DWG
- Secondary: Geosemantics DWG (UML to RDF/OWL/SKOS)

Chapter 1. Introduction

1.1. Scope

This OGC® document specifies enhancements to the application schema profiling functionality provided by ShapeChange.

It specifies rules to convert an application schema in UML to an OWL ontology as well as SKOS concepts and concept schemes.

This OGC® document is applicable to anyone who is defining or implementing an application schema and wants to leverage the advantages of the Model-driven engineering approach, i.e. automatically processing the schema to create schema subsets or to derive components that facilitate implementation.

1.2. Document Contributors

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

Table 1. Contacts

Name	Organization
Ellen D. Badgley	MITRE Corporation
Deborah L. Nichols	MITRE Corporation
Paul Birkel	Geosemantic Resources, LLC
Clemens Portele	interactive instruments GmbH
Johannes Echterhoff (editor)	interactive instruments GmbH

1.3. Future Work

1.3.1. Improve Support for Multilingual Application Schemas

With the new version from 2015, ISO 19109 has a mechanism to provide key information (name/designation, definition, and description) in an application schema in more than one language. RDF supports language tagging of literal values. Combined, this would allow the encoding of descriptive information (definition, description, etc.) in multiple languages when converting an application schema component to RDF. For example, an RDF resource representing a class could have multiple rdfs:label and rdfs:comment properties, documenting the class in different languages. Tools could display the content of an ontology in different languages, according to user preference. From a more general point of view, artifacts automatically derived from the application schema (e.g. documentation in the form of feature catalogues as well as XML Schemas) could be provided in multiple languages, which can facilitate implementation within a multilingual community.

Typically, application schemas are not multilingual yet. The NSG Application Schema, which was

the basis for the work on ShapeChange in OGC Testbed 12, uses English as the primary and only language. Therefore, supporting multilingual application schemas throughout the processing chain of ShapeChange was given a lower priority in Testbed 12.

Future work, however, should incorporate this feature. It would particularly be useful if a multilingual community started developing and using a multilingual application schema.
One reason for doing so would be that the community has multiple official languages (like the International Hydrographic Organization [IHO] and the European Union [EU]), and that artifacts like documentation must be derived from the application schema in multiple languages. Another reason would be that the community wants to support implementers with different linguistic backgrounds. Their work could be facilitated if the application schema - and, subsequently, artifacts derived from it - supported a language in which they are more proficient. This should be of interest for all international communities, like Defense and Intelligence and Aviation.

1.3.2. OWL from OCL Constraints

An application schema may use specific types of OCL constraints that can be mapped to OWL constructs such as class expressions and axioms. For example, an OCL constraint in a subtype can restrict the type of a property inherited from a supertype. This and other types of constraints (e.g. value ranges and allowed values [see the following subsection]) can be represented using OWL property restrictions.

In order to implement such mappings, a detailed analysis of the types of OCL constraints used in application schemas needs to be performed. Then we would need to analyze if and how a specific type of constraint maps to OWL constructs. Ideally, an automated analysis of an OCL expression would identify to which OWL construct a given OCL constraint can be mapped. However, the implementation of such an analysis is non-trivial. Additional knowledge about the types of constraints used in an application schema and the structures of the corresponding OCL expressions can reduce the complexity. For example, the names of OCL constraints could contain a specific code that identifies the type of the constraint.

In general, the derivation of OWL from OCL constraints would result in an ontology that provides more information for automated reasoning.

OWL from OCL to Restrict Allowed Code List Values

The NSG Application Schema (NAS) contains OCL expressions that restrict the use of code values from a code list to a particular subset in the context of a property in a specific class. Consider the example shown in the following figure.

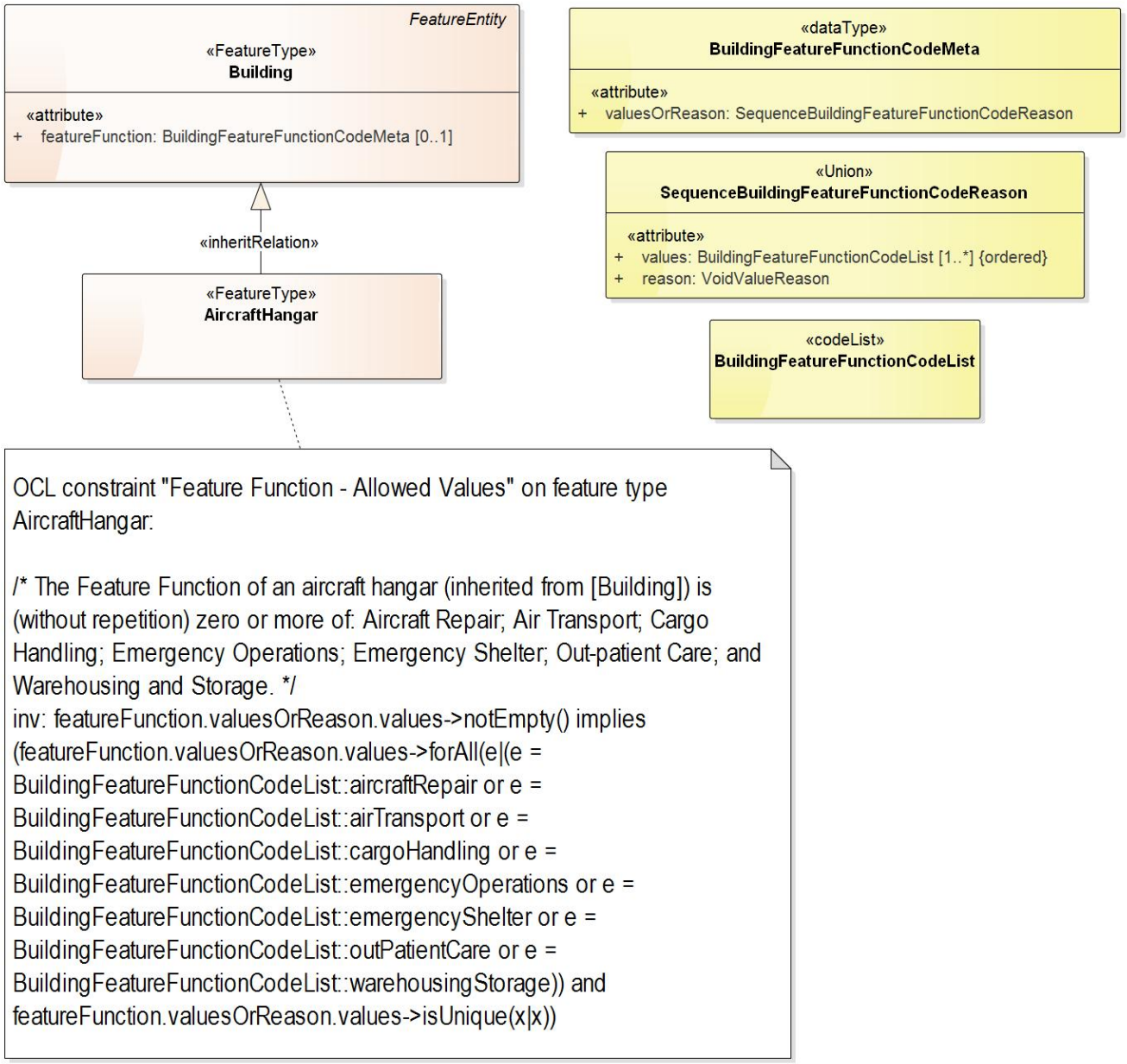

Figure 1. Example of OCL constraint restricting allowed code list values

Here the list of codes from the BuildingFeatureFunctionCodeList is restricted to the set {aircraftRepair, airTransport, cargoHandling, emergencyOperations, emergencyShelter, outPatientCare, warehousingStorage}. The context of the OCL expression is the <<featureType>> AircraftHangar. The restriction ultimately applies to the "values" property in <<Union>> SequenceBuildingFeatureFunctionCodeReason - when used in the context of an AircraftHangar (via the property path of "featureFunction" - inherited from "Building" - and the "valuesOrReason" in <<dataType>> BuildingFeatureFunctionCodeMeta).

NOTE In this example, the "BuildingFeatureFunctionCodeList" is empty, i.e. it does not contain any attributes that would define code values. This is a perfectly legal approach to modelling code lists. It indicates that the code list is managed externally, for example in a registry.

Future work should investigate what is needed to express the semantics of this specific type of OCL constraint in OWL. A conversion rule should be implemented and tested.

A subClassOf expression like the following (here defined in Manchester syntax) on the OWL class representing AircraftHangar could be used to define the constraint, at least the restriction of allowed values:

```
Building.featureFunction only
    (BuildingFeatureFunctionCodeMeta
     and (BuildingFeatureFunctionCodeMeta.values only
{aircraftRepair,airTransport,cargoHandling,emergencyOperations,emergencyShelter,outPat
ientCare,warehousingStorage}))
```

This requires that the individuals representing the code values of BuildingFeatureFunctionCodeList are defined to be different from each other.

Automatically creating this kind of expression requires detailed knowledge about the way the application schema is converted to an ontology. For example, the expression above assumes that the <<Union>> has been converted using *rule-owl-cls-union-replace* defined in Testbed 12. It also assumes that the properties "featureFunction" and "values" are converted with local scope (for further details see the section on locally vs globally scoped properties). Furthermore, there is an assumption that the names of the individuals that represent the code list values are identical to the code list value names found in the OCL expression.

OWL from OCL to Restrict Allowed Geometries

In this case, an OWL representation is needed for the restrictions on geometries that are stated in OCL constraints. NAS OCL constraints may restrict the allowed values for the geometry of a specific type of feature (i.e., some subclass of Feature Entity), to one or two kinds of the three possible geometries. For example, for Runway Intersection, the OCL constraint “Place Representations Allowed” ([Definition] inv: place→forAll(p|p.oclIsKindOf(SurfacePositionInfo))) restricts the value to surface geometry.

NOTE	Geometries in the NAS are represented by subclasses of the abstract class Place Information. Place Information is abstract, as is its subclass Position Information, which has three concrete subclasses: Curve Position Information, Point Position Information, and Surface Position Information.

1.3.3. Validation of RDF Data

Ontologies describe semantics. Reasoning on RDF data can be performed based upon the information found in ontologies. This can lead to new information and knowledge.

Pure validation of RDF data is another use case. Validation can verify that a given dataset is compliant to a specification. This is of interest whenever a data publisher and consumer have agreed to exchange information compliant to a certain specification. This is even more important when there are many publishers and consumers (think of information exchange on and between the communal, regional, national, and international level).

The W3C is developing the Shapes Constraint Language (SHACL)[1]. This could become a key technology for validating RDF data.

Future work should:

- Investigate to which extent SHACL supports validation of RDF data against an application schema ontology, similar to how XML Schema can be used to validate GML data against a GML application schema.
  - An extension of this work would be an investigation to which extent SHACL can also be used to validate JSON-LD data. RDF and JSON-LD are closely aligned - the JSON-LD standard specifies the serialization/deserialization of JSON-LD to/from RDF. JSON-LD could thus be seen as a web-programmer friendly RDF format. Consequently, it should be possible to validate JSON-LD data with SHACL once the JSON data has been serialized to RDF. An example where this would be useful is validation of new data before it is inserted by a transactional Web Feature Service (WFS-T).
- Define and implement rules to automatically derive SHACL descriptions and constraints, called 'shapes', from a UML application schema.
- Test semantic web software components to analyze to which extent they support SHACL.

This work would provide useful and relevant input to decision makers that consider integrating interoperable exchange of RDF - and potentially also JSON-LD - data into their business. Validation is a key aspect in ensuring compliance to interoperability agreements, in particular compliance of data (GML, RDF, and JSON-LD) to application schemas.

NOTE

According to the RDF Data Shapes Working Group Charter the standard was planned to be published as a W3C Recommendation in February 2016. This goal has not been met. The First Public Working Draft was published in October 2015, which is ten months behind schedule according to the charter. This gives an indication that SHACL will not become a W3C Recommendation before early 2017. Depending on when the future work items described above would be carried out, they may need to be performed against a non-final standard.

1.3.4. Ontology for Real-world Objects

The Defence Geospatial Information Working Group (DGIWG) maintains and develops the NATO Geospatial Information Model (NGIM), an application schema for geospatial information. NGIM is part of the NATO Geospatial Information Framework (NGIF). The concepts in NGIM are often rather abstract (e.g. class "Facility"). The NATO Geospatial Real World Object Index (NGRWI) specifies a range of real-world objects that are more user friendly (e.g. "AircraftFactory" and "Embassy"). It does so by specifying which conditions must be met by an NGIM object to classify as a particular real-world object. For example, "AircraftFactory" is an NGIM "Facility" with property "featureFunction" having the value "aircraftManufac". Likewise, "Embassy" is an NGIM "Facility" with property "featureFunction" equal to "embassy".

We used Protégé to briefly test if this kind of condition can be expressed in OWL. The test application schema defines class "Building" and property "religionFacilityType". The real-world object "Cathedral" is defined as a Building with religionFacilityType equal to "Cathedral". The class "Cathedral" can be defined in OWL as follows:

```
rwor:Cathedral a owl:Class ;
    owl:equivalentClass [ a owl:Class ;
            owl:intersectionOf ( ex:Building [ a owl:Restriction ;
                        owl:hasValue "Cathedral" ;
                        owl:onProperty :religionFacilityType ] ) ] .
```

With the inference-aware *Snap SPARQL query* in Protégé it was possible to directly query for all Cathedral real-world objects in a triple store that only contained objects of type Building. In other words, we were able to perform a search by real-world object in a data store that only contained data compliant to the more abstract feature type defined by the application schema.

Benefits: A real-world object index - like NGRWI - can:

- Simplify the identification and retrieval of real-world objects from a data store that contains application schema (e.g. NGIM) compliant feature data.
- This can be useful for building applications around such a data store.
- It can also help implementers that do not have detailed knowledge of all aspects of the application schema. This is especially relevant if the application schema is very large (like NGIM) and supports a wide range of use cases, only a few of which may be of interest to a particular application.

Future work should:

- Perform a complete analysis to which extent the conditions stated in the NGRWI (or its successor) can be expressed in OWL (the test performed in OGC Testbed 12 only looked at one particular example). If the conditions for real-world objects were defined in OCL, this work would be related to the analysis of expressing OCL constraints in OWL.
- Identify the properties that an application schema must have in order to support the creation of a real-world object index (RWI). Analyze if the NSG Application Schema has these properties. If it does, create a draft RWI with a subset of representative real-world objects.
- Analyze how an RWI can be encoded (e.g. in UML or as an Excel workbook) so that a UGAS tool like ShapeChange can read the definitions of real-world objects.
- Define and implement rules to automatically convert the real-world object index into an OWL ontology.
    - This work item would have to take into account how the application schema in UML is converted to OWL. Otherwise, the OWL axioms and class expressions that represent the conditions of real-world objects cannot be implemented through an automated process.

1.3.5. Support Specification in OWL of Properties Re-used Within an Application Schema

The current approach covering ISO 19150-2 rules and ShapeChange capabilities provides for the specification in OWL of UML properties either as globally scoped (that is, applicable to any OWL class) or as scoped locally (that is, defined specifically for the OWL class representing the owning UML class). An additional option is needed to represent the semantics of properties in a UML model

that are used with the same definition in the context of multiple UML classes but which may intentionally not be used with all classes in the model. For example, in the NSG Application Schema (NAS), the property Feature Function is applicable to just the UML classes: Agricultural Colony, Building, Built-up Area, Cableway, Camp, etc. (a long but finite list that excludes many NAS UML classes). Future work should develop transformation rules to represent such “re-used” UML properties by an OWL property whose domain is scoped to a class (using an OWL property domain axiom) that is defined as the union (i.e., ObjectUnionOf) of all the OWL classes representing all the UML classes owning the re-used property. This would require a means of identifying the re-used properties and their owning classes in the application schema (e.g., using a tagged value). An alternative for identifying the re-used properties would be to string-match on identical property names; however, we consider that a less reliable method.

1.3.6. Creating Multiple Ontologies that Support Different Levels of Complexity

When converting an application schema to OWL, ShapeChange currently creates at most five ontologies: one with concepts for all feature, object, and data types, two for the classes representing enumerations and code lists, and two for the individuals derived from enumerations and code lists. These ontologies contain complex class expressions and axioms, for example OWL DisjointClasses and OWL DifferentIndividuals.

Typically, linked data applications only need basic definitions for class and property concepts. The more complex expressions and axioms are mostly required when applying reasoners to infer new information.

Future work should investigate the partitioning of derived ontologies into a) an ontology with the basic concepts, and b) an ontology with the complex expressions and axioms. The latter would import the former. This would result in a simple ontology to be used in linked data applications, and one to be used for cases that require complex reasoning.

1.3.7. Implement the Profiling Workflow Extensions

An application schema specifies the information that is relevant for a community. The information usually supports more than one use case. It is therefore often the case that large schemas - for example the NSG Application Schema (NAS) and the Aeronautical Information Exchange Model (AIXM) - are profiled, so that the resulting schema profile only specifies the information that is relevant to a particular application or use case.

ShapeChange can create application schema profiles, and derive a number of implementation schemas from them, for example XML Schema, Schematron, SQL DDL, and Ontologies.

At the moment, ShapeChange requires that profile information is contained in the application schema itself. Creating, updating, and deleting application schema profiles can become a tedious task. It would be better to keep profile definitions and the application schema separate from each other, and load relevant profiles on-the-fly while executing ShapeChange.

The enhancements and extensions to the profiling workflow designed in Testbed 12 support a clean separation between an application schema and profile definitions. The design should be implemented as part of future activities.

1.3.8. JSON, JSON Schema, and JSON-LD

Work in Testbed 12 included a brief investigation of automatically converting the NAS to JSON. The main focus of this investigation was to test the conversion to JSON Schema with the current NAS. The conversion to JSON Schema was added to ShapeChange in Testbed 9 (for further details, see the OWS-9 SSI UGAS ER). Another aspect of the investigation was the automatic derivation of JSON-LD context documents. The observations and recommendations for future work that resulted from these investigations are documented in the following subsections.

Converting the NAS to JSON Schema

1. Association classes can be transformed as defined by GML 3.3. A generic transformer has been implemented in Testbed 12 to support the conversion of association classes when deriving an ontology from an application schema. The transformer can be re-used in any workflow that requires the mapping of association classes.
2. The current implementation of the conversion to JSON Schema produces a schema that conforms to JSON Schema, draft v3. Enhancements available in JSON Schema, draft v4, should be analyzed to identify if they would improve the conversion of application schemas to JSON Schema.
    a. One example where draft v4 looks promising is improved support for inheritance, using the keyword "allOf" (for further details on this keyword, see Understanding JSON Schema or json-schema-validation).
    b. In the NAS, the subtypes of *PositionInfo* are used to represent geometry information in combination with additional information about the place represented by the geometry. The JSON Schema target does not support this modelling approach yet, since the geometry types from ISO 19107 are not modelled as property value types, but as supertypes of the NAS PositionInfo subtypes. Inheritance flattening is not appropriate in this case. However, if JSON Schemas for geometry types were defined, as recommended by the OWS-9 SSI UGAS ER, then the PositionInfo subtypes could be implemented using combinations of JSON Schemas with the "allOf" keyword. Note that an official JSON Schema for GeoJSON does not exist yet, although some work has been done in that regard.
3. JSON Schemas should be defined for ISO schemas - or profiles of those schemas, for example ISO 19107, ISO 19108, ISO 19115, and ISO 19157. A common set of JSON Schemas that implement types specified in the ISO schemas would facilitate the implementation of application schemas such as the NAS on platforms that use JSON encoded data. Standardized ISO JSON Schemas would provide a basis to specify a JSON encoding for features according to ISO 19109 in a consistent, testable way.
4. The ShapeChange target that converts an application schema into a JSON Schema has been developed in OGC Testbed 9. Since then, a number of generic model transformation capabilities were added to ShapeChange. One of them, the "flattener", can simplify a conceptual model, similar to what is necessary for the conversion to JSON Schema. A future revision of the existing JSON Schema target should leverage these generic transformation capabilities, for example the transformation of properties with a multiplicity greater than one.
5. The conversion to JSON Schema currently ignores mixin types and basic types. For further details on these types, see clauses Mixin Types and Basic Types. Support for these special types

should be added, to cover all types that occur in the NAS.

Conversion to JSON-LD Context Documents

JSON-LD is a lightweight syntax to serialize Linked Data in JSON. As the name suggests, it supports linking in and between datasets, much like Xlinks in GML data.

A key aspect that JSON-LD adds to JSON is semantic tagging of data elements. Like in XML, each element can be assigned to a namespace. This allows clients to identify the exact meaning of an element, especially if there are multiple elements that happen to have the same name. JSON-LD context documents are used to provide the necessary information. A context document typically references terms from one or more vocabulary or ontology. These terms then provide the semantics of elements in the actual JSON data.

Being able to identify the meaning of a data element, especially the namespace it belongs to, is important in more complex environments. For example, an application schema may import multiple other schemas, some of which can be extended dynamically. Actual JSON data would then include elements defined in multiple namespaces. If these elements had the same name, clients would not know the exact meaning of the elements - unless that meaning was defined through JSON-LD. Clients that can identify the meaning of a data element can parse it correctly.

The serialization of an ontology in JSON-LD does not result in the JSON-LD context documents that would be needed for use in actual JSON-LD data. However, it would be possible to convert an application schema to JSON-LD context documents. This can be done while converting the schema to RDF/OWL/SKOS (which is documented in this report).

Validation of JSON and JSON-LD Data

Creating a JSON Schema serves two purposes:

1. validating JSON data, where needed, and
2. facilitating the development and setup of software through
    a. automatically deriving programming language specific bindings from the JSON Schema (with tools like jsonschema2pojo and NJsonSchema for .NET)
    b. configuring a service with JSON Schemas to specify the data that the service shall consume or provide (for example, the Open API specification).

A question that came up in OGC Testbed 12 was if JSON Schema could be used to validate JSON-LD data. To answer this question, further analysis is required. However, there is at least one issue: JSON-LD has multiple ways to encode the same information. JSON-LD Framing could be one approach to solve this issue:

> A JSON-LD document is a representation of a directed graph. A single directed graph can have many different serializations, each expressing exactly the same information. Developers typically work with trees, represented as JSON objects. While mapping a graph to a tree can be done, the layout of the end result must be specified in advance. A Frame can be used by a developer on a JSON-LD document to specify a deterministic layout for a graph.
>
> — From the introduction of JSON-LD Framing

With framing, a JSON-LD document could be brought into a specific layout, which could then be validated using a JSON Schema. Further analysis and testing would be necessary to verify that this approach is feasible.

There could also be another solution for validating JSON-LD data. If the data transformed well to RDF (this requires further analysis), then the Shapes Constraint Language (SHACL)[1] - which is also mentioned in the future work item on Validation of RDF Data - might be better suited to validate the data. SHACL would be processing the pure RDF data, and be independent of the data format.

1.4. Changes to the OGC Standards Baseline

This report documents new developments in the area of application schema profiling and conversion of application schemas in UML to RDF/OWL/SKOS. The work did not identify a need for change requests against the OGC standards baseline.

1.5. 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 2. References

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

- ISO 19103:2015, Geographic information - Conceptual schema language
- ISO 19109:2015, Geographic information - Rules for application schema
- ISO 19150-2:2015, Geographic information — Ontology — Part 2: Rules for developing ontologies in the Web Ontology Language (OWL).
- ISO/IEC 19505-2:2012, Information technology - Object Management Group Unified Modeling Language (OMG UML), Superstructure
- OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012, available online at http://www.w3.org/TR/owl2-syntax/
- OGC Geography Markup Language (GML) - Extended schemas and encoding rules, OGC document number 10-129r1

Chapter 3. Abbreviated Terms

AIXM	Aeronautical Information Exchange Model
DDL	Data Definition Language
DGIWG	Defence Geospatial Information Working Group
DIS	Draft International Standard
DWG	Domain Working Group
ER	Engineering Report
EU	European Union
FOL	First Order Logic
FPS	Feature Portrayal Service
GML	Geography Markup Language
IHO	International Hydrographic Organization
IRI	Internationalized Resource Identifier
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
MDE	Model-driven engineering
NAS	NSG Application Schema
NEO	NSG Enterprise Ontology
NGIF	NATO Geospatial Information Framework
NGIM	NATO Geospatial Information Model
NGRWI	NATO Geospatial Real World Object Index
NSG	U.S. National System for Geospatial Intelligence
NTAX	NSG Taxonomy
OCL	Object Constraint Language
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
RWI	Real-world Object Index
SBVR	Semantics of Business Vocabulary and Business Rules

SDI	Spatial Data Infrastructure
SHACL	Shapes Constraint Language
SKOS	Simple Knowledge Organization System
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
Turtle	Terse RDF Triple Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WFS	Web Feature Service
WFS-T	Web Feature Service - Transactional
WMS	Web Map Service
XMI	XML Metadata Interchange
XML	Extensible Markup Language

Chapter 4. Overview

An application schema is a fundamental building block of interoperable spatial data infrastructures (SDI). It is a conceptual model that defines the structure and content of geographic information.

Through automated processing, an application schema can be converted into an implementation schema. Such a schema describes the content and structure of geographic information in a technology-specific way, for example XML schemas, Schematron schemas, database schemas, JSON schemas, and ontologies. Developers use these schemas and artifacts to write applications. In general, creating and using application schemas is a realization of the Model-driven engineering (MDE) approach.

This report documents two areas of processing an application schema that have been enhanced in OGC Testbed 12:

- Creating an application schema profile to support implementations that focus on a subset of the use cases in scope of the original application schema - see chapter Profiling.
- Converting an application schema to an ontology (using RDF(S)/OWL/SKOS) to support Semantic Web and Linked Data implementations - see chapter UML to RDF/OWL/SKOS.

NOTE	Converting an application schema to JSON Schema and JSON-LD Context documents to support mobile and browser based applications has been investigated as well, but only to a very limited extent. The results constitute a future work item.

The remainder of this chapter is structured as follows: The next subsection highlights the value of application schemas for the geospatial community. The following subsection describes the capabilities of ShapeChange, a tool for processing application schemas. ShapeChange was used to implement the enhancements in application schema processing developed in OGC Testbed 12.

4.1. The Value of Application Schemas

An application schema defines structure and semantics of geographic features for a specific domain, community, or application.

ISO 19109 defines rules for writing an application schema, while ISO 19136 defines encoding rules to map an ISO 19109 conformant application schema (using UML as the conceptual schema language [1: ISO 19103 identifies the Unified Modeling Language (UML) and the Object Constraint Language (OCL) as the conceptual schema language for specification of geographic information.]) into a GML application schema (which is an XML Schema).

Web Feature Services (WFS) provide interoperable transactions on and access to geographic features. The types of these features are defined in application schemas. By default, each WFS supports features described using a GML application schema. Additional encodings are allowed, too. The geographic information provided by a WFS can be used by various components within a Spatial Data Infrastructure (SDI) to support a multitude of use cases. A common use case, for example, is to visualize the information using components like the Web Map Service (WMS) and Feature Portrayal Service (FPS).

Summary: application schemas are a key enabler and building block of geographic information systems.

4.2. ShapeChange

ShapeChange is a software tool to process application schemas. The following figure illustrates how it works.

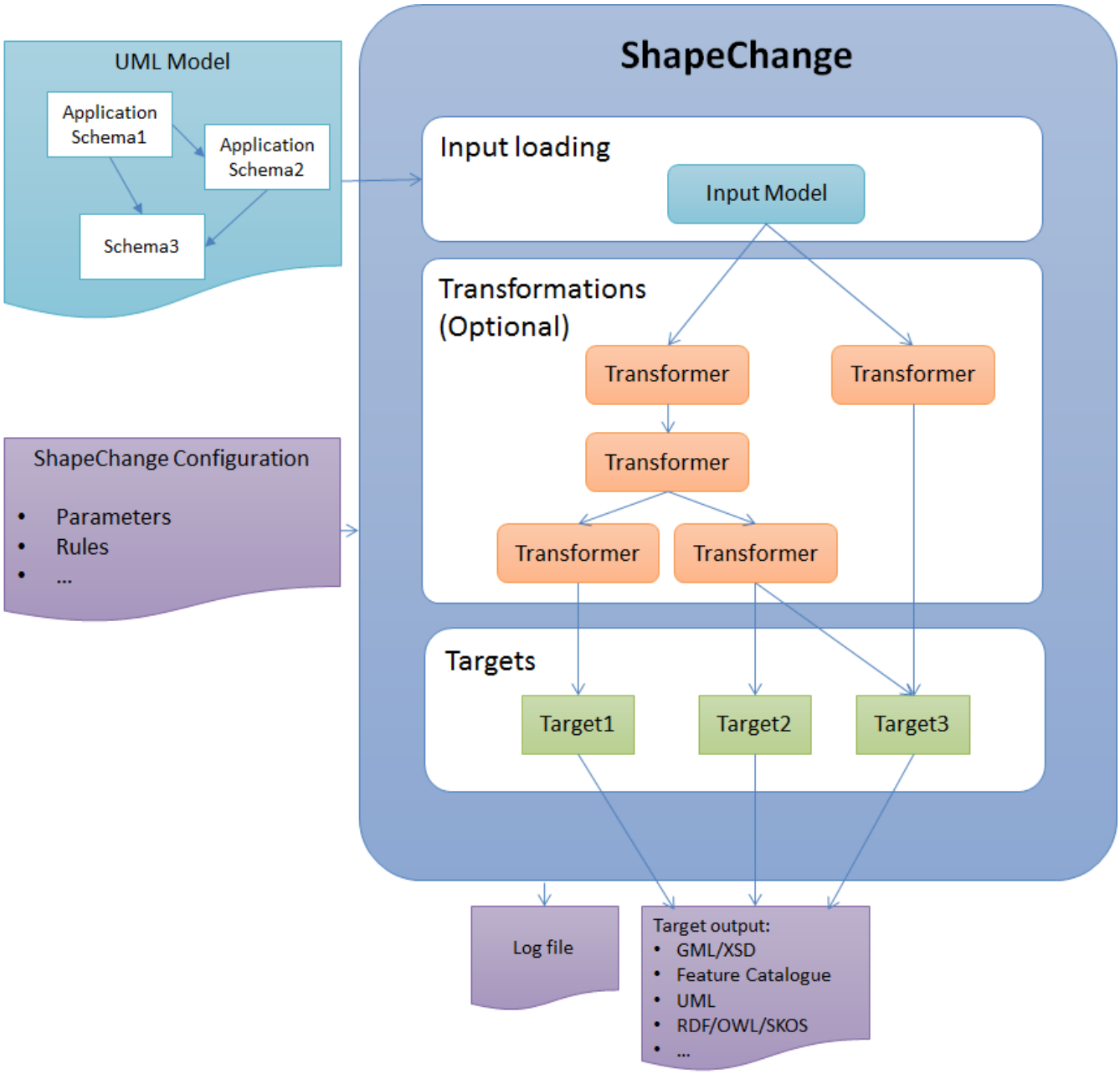

Figure 2. ShapeChange - Overview

- A UML model with one or more application schemas is loaded. The model can be loaded from an Enterprise Architect repository, an XMI file, or a NAS-conformant database. At least one of the schemas is selected (via the configuration) for further processing.
  - NOTE: By default, ShapeChange processes all application schemas it finds in the configuration. Typically, parameters are used to select a subset of all schemas contained the model. Schemas can be selected by name as well as a regular expression to match the name or the target namespace.

- The model can be transformed multiple times.
- The input model as well as any transformed model is encoded by one or more targets. Outputs created by ShapeChange include, but are not limited to:
  - XML Schemas
  - Feature catalogues (documentation, in different formats)
  - Ontologies (RDF, OWL, SKOS)
  - ArcGIS Workspace models
  - SQL DDL
  - JSON Schemas
- Process execution is documented in a log file. Issues encountered when loading the input model, or when processing it, will be reported in this file.
- A configuration file defines parameters and rules for loading the input model as well as executing transformations and targets.

The behavior for processing a model is primarily controlled through conversion rules. In addition, parameters and other configuration elements (like map entries) influence the way a schema is processed.

An *encoding rule* is a specific set of conversion rules. The encoding rule can be defined in a standard (like ISO 19136). However, with ShapeChange, a community can also define its own encoding rule.

More detailed documentation is available on http://shapechange.net

Chapter 5. Status Quo and New Requirements Statement

This chapter explains the status quo and the new requirements or existing problems and issues that have been addressed by this ER.

5.1. Status Quo

5.1.1. Profiling

Application schemas as defined by ISO 19109 are a key building block in Spatial Data Infrastructures (SDI). They define the structure and semantics of geospatial features that are of interest to a specific domain. Domains like defense and aviation have large and complex schemas that support a wide range of applications and use cases. A given application usually only depends on a subset of such an overarching schema. In other words, it depends on a schema profile.

ShapeChange processes ISO 19109 compliant application schemas. One of the available processing steps is the *Profiler* transformation, with which a model profile can be created. The *Profiler* can remove classes and their properties from a schema if they do not belong to one or more specific profiles (indicated via information contained in the tagged value "profiles"). The resulting profile can be processed like any normal application schema. It can be written as a GML application schema / XML Schema, a feature catalogue, or a UML model.

The behavior for handling constraints while profiling is rather limited. Constraints can either be kept as-is or be removed. When removing constraints, either all constraints are removed or only those constraints are removed whose name contains the name of a property that has been removed.

The *Profiler* can also perform some pre- and postprocessing around the actual work of creating a model profile. During preprocessing, it can check the profile settings of model elements for consistency. During postprocessing, it can remove residual types (all non-featuretype classes that are not used directly or indirectly by feature types of the model) and empty packages.

5.1.2. UML to RDF/OWL/SKOS

The first ShapeChange target to automatically map an ISO 19109 compliant application schema in UML to an ontology representation (in RDF, OWL, and SKOS) was created in OGC Testbed 8 in 2011 (for further details, see [4] - section 8.1).

At the same time, the ISO TC 211 project 19150 was already active, working on two standards: ISO 19150-1 and ISO 19150-2. ISO 19150-1 was published in 2012 and defines a framework for semantic interoperability of geographic information. The development of ISO 19150-2 was completed in 2015, with ISO/DIS (a draft version) published in 2013. ISO 19150-2 defines rules and guidelines for the development of OWL ontologies for ISO geographic information UML models and application schemas.

In 2014, a new ShapeChange target was created based upon ISO/DIS 19150-2.

5.2. Requirements Statement

5.2.1. Profiling

The profiling functionality shall be enhanced to:

1. Support restrictions on model elements, and
2. Enable restrictions to be specified by an external configuration file rather than being carried exclusively as UML tag-values in the application schema itself.

The following sections describe the requirements in more detail.

Adding Additional Options for Restricting Model Elements

The following restrictions shall be supported:

A. Overwrite the multiplicity of properties, e.g.:
   a. 1..* → 1..n or 1
   b. 0..1 → 1
   c. 0..* → 0..n or 0..1

B. Adjust uniqueness and ordering of properties:
   a. Adjust the uniqueness of values by setting “isUnique” to true or false;
   b. Adjust the ordering of values by setting “isOrdered” to true or false.

C. Set association role navigability to true or false.

D. Make classes non-instantiable/abstract if, for example, their instantiation is disallowed by the profile, but subclass instantiation is allowed.

E. Adjust OCL constraints:
   a. Support the addition of new OCL constraints to further constrain classes and properties selected for inclusion in the profile.
   b. Support overwriting a constraint with a more restrictive one specified for a profile.
   c. Support the deletion of constraints in a profile.
   d. Support the parsing/validation of constraints with a profile as context (to ensure that the constraint does not rely on model elements that may have been removed through profiling).

Specifying Profile Restrictions by an External Configuration File

This represents a paradigm shift regarding how profile information may be conveyed to the *Profiler* transformation. Whereas ShapeChange currently expects profile information to be provided in the input model, the enhanced version of ShapeChange would be able to load profile information from an external file. This would support workflows where profiles are developed "on-the-fly" against a stable, base application schema, which then does not need to be modified during profile development.

The external configuration file should be in a format that is easy to develop and maintain. It must be possible to overwrite all profile information that may be contained in the input model with the information of the external configuration file.

5.2.2. UML to RDF/SKOS/OWL

The general requirements for the conversion of an application schema in UML to RDF/SKOS/OWL in Testbed 12 were defined as follows:

- Extend the ShapeChange output target for RDF/OWL based on additional/alternative rules from those specified in ISO 19150-2 to improve interoperability with, and reuse of, ontologies from outside the geospatial community. The additional/alternative rules are defined during the Testbed. Add support for N-Triples format.
- Add ShapeChange output target for RDF/SKOS to support specification of Controlled Vocabularies and Taxonomies, including support for N-Triples format.

The existing ShapeChange targets to derive an OWL ontology from an application schema in UML do not implement the rules of the final version of ISO 19150-2. Therefore, the ShapeChange target that is based on ISO/DIS 19150-2 needs to be updated to implement the rules from the final version of ISO 19150-2.

NOTE	An analysis of the conversion rules from the draft and final version of ISO 19150-2 is documented in Annex A.

Chapter 6. Solutions

6.1. Targeted Solutions

6.1.1. Profiling

Loading Profile Information

Two options were identified for loading profile information from an external source:

- Let the Profiler transformation load the information.
- Load the information in a separate transformation.

6.1.2. Profile Configuration Format

XML and Excel have been discussed as formats for encoding profile information in an external file.

To dynamically support new formats, the ShapeChange configuration should support the specification of the class that implements profile loading.

6.1.3. Intelligent Automated Revision of Constraints vs. Simple Validation

If profiling removes properties and classes from the model that are required by OCL constraints, then ShapeChange should at least report which constraints have become invalid due to profiling. Ideally, constraints can be updated during or after profiling, to take into account the model changes in the model that were introduced by this transformation.

6.1.4. UML to RDF/SKOS/OWL

One option to automatically derive an OWL ontology from an application schema in UML would have been to use the rules defined in the final version of ISO 19150-2 as-is. However, a detailed analysis of these rules showed that - in some cases - they are:

- vague - for example regarding the content of the “iso19150-2:constraint” annotation,
- potentially insufficient - for example regarding the encoding of unions (see below), or
- silent - for example regarding the conversion of association classes.

Therefore, another option was identified: Use the rules from ISO 19150-2 as a basis, and adapt or add to them as necessary.

The following subsections document some of the issues with the conversion rules of ISO 19150-2 in more detail.

Conversion of Unions

ISO 19150-2 provides an encoding for union types. The union is implemented as an OWL class that uses an ObjectUnionOf (a union of class expressions - in RDF syntax: owl:unionOf) to represent the

members of the union. This approach is insufficient, for the following reasons:

- It does not handle cases where the value types of union properties are a mix of object and datatypes.
- It does not handle cases where the same value type is used by more than one union property.
- It focuses on the value types of the union properties, completely ignoring the fact that the properties themselves can carry meaning and therefore must not be discarded.
- It does not take into account that a union property can have multiplicity other than exactly 1.

An alternative approach must be found that solves these issues.

Conversion of Generalization/Inheritance Relationship - Disjointness

ISO 19150-2 states that a UML generalization/inheritance relationship shall be implemented as an rdfs:subClassOf declaration. It does not address the representation of uniqueness between subtypes.

ISO 19109 covers uniqueness of subtypes via the *uniqueInstance* attribute of the *InheritanceRelation* <<metaclass>>. The definition of *uniqueInstance* in ISO 19109 is: *"UniqueInstance is a Boolean variable, where TRUE means that an instance of the supertype shall not be an instance of more than one of the subtypes, whereas FALSE means that an instance of the supertype may be an instance of more than one subtype. Optional with a default value of 'true'"*.

NOTE	Application schemas typically do not limit generalization/inheritance relationships to feature types. The NAS, for example, also uses this relationship between <<type>> classifiers.

In ISO 19109 and UML, the default value for *uniqueInstance* in an inheritance relationship is 'true'. Given the example shown in the following figure, uniqueInstance=true says that an *Animal* can be an *Elephant* or a *Mosquito*, but not both at the same time. In other words, the subtypes are mutually disjoint.

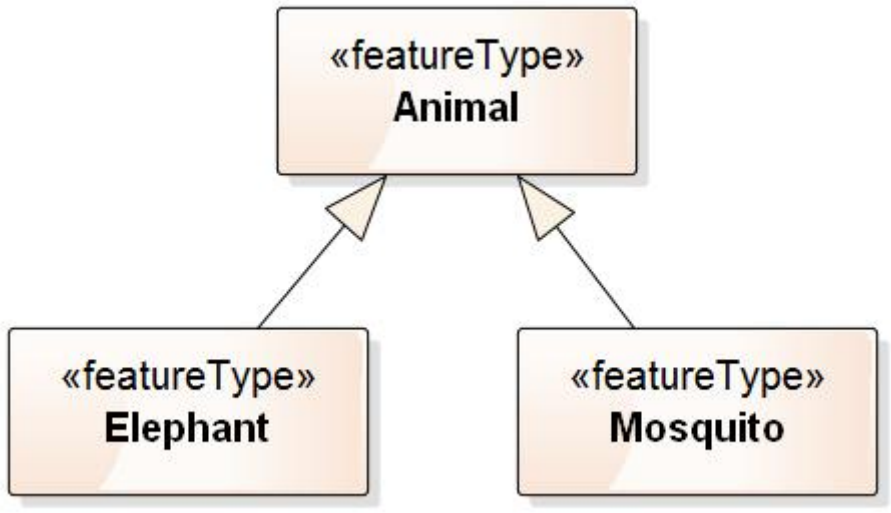

Figure 3. Inheritance example to illustrate uniqueness

NOTE	In the NAS (Part 1, Version 7.0) *uniqueInstance* is always TRUE.

In the W3C Web Ontology Language (OWL), subtypes are not assumed to be mutually disjoint. The OWL 2 Primer observes that "…the information that two classes are disjoint is part of our background knowledge and has to be explicitly stated for a reasoning system to make use of it." [3].

The disjointness constraints imposed by a TRUE value of uniqueInstance are therefore not part of the meaning of rdfs:subClassOf. Given the example, that means that an individual *Animal* may be both an *Elephant* and a *Mosquito*.

An additional conversion rule is required to address the representation of uniqueness/disjointness of subtypes in inheritance trees.

6.2. Recommendations

6.2.1. Profiling

Loading Profile Information

Instead of adding the functionality to load profile information from an external file to the *Profiler* transformation, we assigned this functionality to two transformations:

- The *ProfileLoader* is a new transformation that loads information from an external source to populate the "profiles" tagged value of model elements.
- The *ConstraintLoader* has been developed in Testbed 11 and is an ideal candidate to host all functionality related to the loading of constraints from an external source.

The responsibilities for loading profile information and of actually creating a profile are thus clearly separated. Even though this results in an additional processing step, represented by an additional 'Transformer' in the ShapeChange configuration file, it avoids a 'scope creep' for the Profiler.

Profile Configuration Format

It was decided that XML would suffice to store profile information.

Intelligent and Automated Revision of Constraints vs. Simple Validation

Early on in the Testbed it was decided that developing a specification for an automated revision of OCL constraints, based upon information from profiles, would be impossible to achieve during the Testbed without putting all other requirements documented in this report at risk.

A simple validation of constraints after a transformation was pursued. This functionality already provides much needed support to a domain expert who defines application schema profiles. The validation report will inform the expert which of the original constraints from the application schema need to be removed or replaced when creating a particular profile of the schema.

6.2.2. UML to RDF/SKOS/OWL

Early on in the Testbed, it became apparent that simply relying only on the rules defined by ISO 19150-2 to convert an application schema to an OWL ontology did not meet the requirements for deriving an OWL ontology from the NSG Application Schema (NAS).

Therefore, it was recommended to introduce a range of additional conversion rules and configuration options as well as adaptations of the rules from ISO 19150-2. This solution is documented in chapter UML to RDF/OWL/SKOS. It addresses the representation of union semantics as well as the representation of uniqueness/disjointness of subtypes in inheritance trees.

Chapter 7. Profiling

This chapter discusses the enhancements to the profiling capability of ShapeChange.

NOTE	The profiling functionality that is already implemented by ShapeChange is documented at http://shapechange.net/transformations/profiler

7.1. Overview

Some application schemas - for example in the defense and intelligence as well as aviation domains - are designed to support a range of applications and use cases. Often, software components are built to support a specific subset of these use cases. Consequently, only a subset of the whole application schema is of interest to such components. This is where profiling comes into play: profiling can remove and update model elements (classes, properties, constraints as well as their details) so that the resulting model is tailored for a particular application, and all aspects that are irrelevant for the application have been removed or suppressed.

Manually creating a schema profile can be a tedious, error prone process. Therefore, ShapeChange automates this process. A domain expert can focus on specifying profile definitions. ShapeChange reads these profile definitions and then creates the schema profiles. Various outputs can then be derived for each schema profile (for further details, see the overview of ShapeChange).

The requirements on the profiling mechanism are documented in section Profiling. The following sections describe the design of ShapeChange to support these requirements.

The overall workflow of profiling a model is depicted in the next figure.

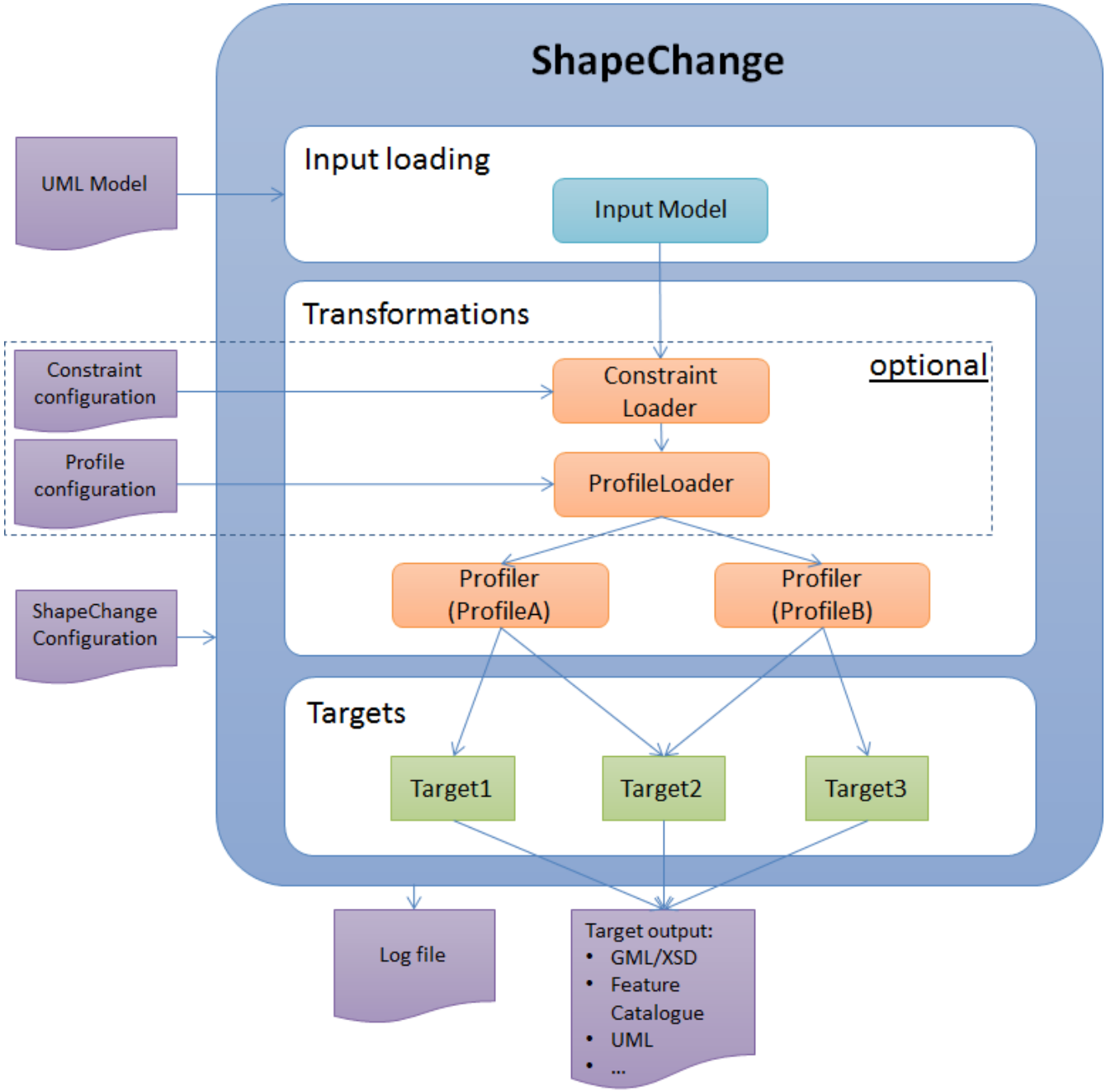

Figure 4. Profiling workflow

7.2. Representation of Profile Information

Three types of information are relevant for profiling:

- *profile identifiers*
- *profile constraints*
- key-value-pair formatted *profile metadata*

7.2.1. Profile Identifier

A *profile identifier* consists of a name (for the profile definition) and optional version indicator (which is not required in Testbed 12). Each model element (class, property, constraint) can be assigned to one or more profile definitions through profile identifiers.

For classes and properties, profile identifiers are contained in the tagged value "profiles".

Constraints do not have tagged values. That is why we define a specific constraint type, a *profile constraint*, which includes profile identifier(s) at the start of the constraint text.

7.2.2. Profile Constraint

A constraint is defined by its name, the model element it belongs to, a constraint text (that can include comments), and the constraint type [2: The status of the constraint usually is irrelevant].

If the type of the constraint is "Profile" (ignoring case) then ShapeChange will recognize it as a constraint that has one or more profile identifiers in the first line of the constraint text. If the profile constraint is used to indicate that a particular constraint (most likely defined in the input model for the original schema) shall be deleted then the remaining text will be empty. Otherwise, the second line of text specifies the actual constraint type (e.g. "OCL" or "Invariant"). In that case, the actual constraint text shall not be empty (so there must be at least three lines of text, because it represents the actual constraint text.

7.2.3. Profile Metadata

Simple metadata can be provided for profiles in the form of key-value-pairs.

This metadata can be added to a profile identifier as follows:

- ProfileA(multiplicity[1..n]),ProfileB(multiplicity[1])
- ProfileA(isAbstract[true])
- ProfileA(isAbstract[true],geometry[P,S])
- ProfileA[4-](isAbstract[true]),ProfileB(isAbstract[true])

A profile identifier would then have the following structure:

1. profile name
2. version indicator (optional)
3. profile metadata (optional)

NOTE	Profile metadata cannot be specified for specific versions of a profile (which can be specified via the optional version indicator of the profile identifier). This would be overly complicated and is not a use case for Testbed 12.

7.3. Profiling Processing Steps

The Profiling workflow shows that three types of model transformations are used to realize the desired profiling. They are described in the following sections.

7.3.1. ConstraintLoader

This transformation loads constraints from an external source (supported format is XML and Excel)

and adds them to the model. More specifically: it adds constraints only to the schemas that have been selected for processing.

The following table describes the information that is provided for each constraint

Table 2. ConstraintLoader - Constraint Information

Information Item	Type	Required / Optional	Default Value	Description
Constraint Name	String	Optional	-	Name of the Constraint. If no value is provided, the ConstraintLoader will attempt to extract the name from the constraint expression based upon the following structure (in BNF): [<name> ':'] <constraint expression>.
Constraint Type	String	Optional	-	The type of the constraint, for example "Invariant", "OCL", "SBVR", "Profile". If no value is provided then the ConstraintLoader uses the value provided via the configuration parameter "defaultConstraintType". If that parameter is not provided, then the type is left empty.
Constraint Expression	String	Required	-	Specifies the constraint, using the following structure (in BNF): *[<name> ':'] <constraint expression>*.
Comment	String	Optional	-	Describes the constraint.
Schema Package Name	String	Required	-	Name of the schema package that contains the class or property a constraint is specified for.
Context Element Name	String	Required	-	Name of the schema class or property a constraint is defined for. If the context element type is "Property" then the context element name has the following structure: <class name>'::'<property name>.
Context Element Type	Enumeration: Class & Property	Optional	Class	The type of the element a constraint is defined for. This can either be a class or property.
Profiles	String	Optional	-	Comma-separated list of profile identifiers. Can be used to define a constraint for a specific set of profiles.

Constraint information can be provided in different formats. XML (the XML Schema is documented in Annex A) and Excel are supported.

The following listing gives an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Constraints xmlns="http://shapechange.net/constraintLoader/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://shapechange.net/constraintLoader/1.0
./constraintLoader.xsd">
  <constraint>
    <Constraint>
      <constraintName>Place Representations Allowed</constraintName>
      <constraintType>OCL</constraintType>
      <constraintExpression>inv: place-
>forAll(p|(p.oclIsKindOf(PointPositionInfo)))</constraintExpression>
      <schemaPackageName>Schema A</schemaPackageName>
      <contextElementName>WindTurbine</contextElementName>
      <profile>
        <ProfileIdentifier>
          <name>PlaceGeomTestV3</name>
        </ProfileIdentifier>
      </profile>
    </Constraint>
  </constraint>
  <constraint>
    <Constraint>
      <constraintName>Place Representations Allowed</constraintName>
      <constraintType>OCL</constraintType>
      <constraintExpression>inv: place-
>forAll(p|(p.oclIsKindOf(PointPositionInfo)))</constraintExpression>
      <schemaPackageName>Schema A</schemaPackageName>
      <contextElementName>WindFarm</contextElementName>
      <profile>
        <ProfileIdentifier>
          <name>PlaceGeomTestV3</name>
        </ProfileIdentifier>
      </profile>
    </Constraint>
  </constraint>
  <constraint>
    <Constraint>
      <constraintName>Place Representations Allowed</constraintName>
      <constraintType>OCL</constraintType>
      <constraintExpression>inv: place-
>forAll(p|(p.oclIsKindOf(PointPositionInfo)))</constraintExpression>
      <schemaPackageName>Schema A</schemaPackageName>
      <contextElementName>Zoo</contextElementName>
      <profile>
        <ProfileIdentifier>
          <name>PlaceGeomTestV3</name>
        </ProfileIdentifier>
      </profile>
    </Constraint>
  </constraint>
  <constraint>
    <Constraint>
      <constraintName>Place Representations Allowed</constraintName>
      <constraintType>OCL</constraintType>
```

```
      <constraintExpression>inv: place-
>forAll(p|(p.oclIsKindOf(PointPositionInfo)))</constraintExpression>
      <schemaPackageName>Schema A</schemaPackageName>
      <contextElementName>Windmill</contextElementName>
      <profile>
        <ProfileIdentifier>
          <name>PlaceGeomTestV3</name>
        </ProfileIdentifier>
      </profile>
    </Constraint>
  </constraint>
  <constraint>
    <Constraint>
      <constraintName>Place Representations Allowed</constraintName>
      <constraintType>OCL</constraintType>
      <constraintExpression>inv: place-
>forAll(p|(p.oclIsKindOf(PointPositionInfo)))</constraintExpression>
      <schemaPackageName>Schema A</schemaPackageName>
      <contextElementName>Wreck</contextElementName>
      <profile>
        <ProfileIdentifier>
          <name>PlaceGeomTestV3</name>
        </ProfileIdentifier>
      </profile>
    </Constraint>
  </constraint>
</Constraints>
```

The parameters in the following table are used to provide information required by the transformation process.

Table 3. ConstraintLoader - Parameters

Parameter Name	Type	Required / Optional	Default Value	Description
inputFile	String	Required	-	Location of the file with constraint information. This can be a file path (absolute or relative to the working directory) and URL.
defaultConstraintType	String	Optional	-	If a value is provided, the ConstraintLoader will use it as the constraint type whenever a constraint from the input does not define the type itself.
oclConstraintTypeRegex	String	Optional	-	Regular expression that is applied to the type of a constraint. If the type matches, then the constraint is treated as an OCL constraint. If the parameter is not present or an empty value is given, then a constraint will not be recognized as OCL constraint. Also see NOTE1 and NOTE2 at the bottom of this table.

Parameter Name	Type	Required / Optional	Default Value	Description
folConstraintTypeRegex	String	Optional	-	Regular expression that is applied to the type of a constraint. If the type matches, then the constraint is treated as a First Order Logic (FOL) constraint. If the parameter is not present or an empty value is given, then a constraint will not be recognized as FOL constraint. Also see NOTE1 and NOTE2 at the bottom of this table.
profileConstraintTypeRegex	String	Optional	-	Regular expression that is applied to the type of a constraint. If the type matches, then the constraint is treated as a profile constraint. If the parameter is not present or an empty value is given, then a constraint will not be recognized as profile constraint. Also see NOTE1 and NOTE2 at the bottom of this table.
NOTE1: A constraints whose type is not recognized will be converted to a text constraint. NOTE2: the constraint type regexes are evaluated with following priority: profileConstraintTypeRegex > oclConstraintTypeRegex > folConstraintTypeRegex				

By default, the ConstraintLoader adds the constraints found in the input to the constraint set of their respective context elements. Previously existing constraints are not changed (e.g. overwritten or deleted). The following rules can be used to modify the default behavior.

Table 4. ConstraintLoader - Rules

Rule ID	Parameters	Map Entries	Behavior
rule-trf-constraintloader-deleteAllPreexistingConstraints	-	-	The ConstraintLoader deletes all constraints in the input model (more specifically: in the schemas selected for processing) before loading the constraints from the input file. NOTE: this rule has higher priority than rule-trf-constraintloader-overwritePreexistingNonProfileConstraint
rule-trf-constraintloader-overwritePreexistingNonProfileConstraint	-	-	The ConstraintLoader overwrites a preexisting constraint if the input file contains a constraint with the same name, same context element, and both constraints are not profile constraints. If the constraint from the input file has an empty expression then the preexisting constraint will be deleted.

7.3.2. ProfileLoader

This transformation populates the profiles tagged value of classes and properties with profile information provided via an input file. It does not add new constraints - that functionality is covered by the ConstraintLoader.

Profile information can be provided in XML (the XML Schema is documented in Annex C). The following listing gives an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ProfileInformation xmlns="http://shapechange.net/profileLoader/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://shapechange.net/profileLoader/1.0 ./profileLoader.xsd">
  <schema>
    <Schema>
      <packageName>Schema A</packageName>
      <requirement>
        <Requirement>
          <class>FeatureEntity</class>
          <profile>
            <Profile>
              <identifier>
                <ProfileIdentifier>
                  <name>PlaceGeomTestV3</name>
                </ProfileIdentifier>
              </identifier>
            </Profile>
          </profile>
          <profile>
            <Profile>
              <identifier>
                <ProfileIdentifier>
                  <name>NameOfAnotherProfile</name>
                  <versionIndicator>2.2-</versionIndicator>
                </ProfileIdentifier>
              </identifier>
            </Profile>
          </profile>
        </Requirement>
      </requirement>
      <requirement>
        <Requirement>
          <class>WindTurbine</class>
          <profile>
            <Profile>
              <identifier>
                <ProfileIdentifier>
                  <name>PlaceGeomTestV3</name>
                </ProfileIdentifier>
              </identifier>
            </Profile>
          </profile>
        </Requirement>
      </requirement>
      <requirement>
        <Requirement>
```

```
          <class>WindTurbine</class>
          <property>windFarm</property>
          <profile>
            <Profile>
              <identifier>
                <ProfileIdentifier>
                  <name>PlaceGeomTestV3</name>
                </ProfileIdentifier>
              </identifier>
              <metadata>
                <KeyValuePair>
                  <key>multiplicity</key>
                  <value>1</value>
                </KeyValuePair>
              </metadata>
            </Profile>
          </profile>
        </Requirement>
      </requirement>
.
.
.
      <requirement>
        <Requirement>
          <class>Wetland</class>
          <profile>
            <Profile>
              <identifier>
                <ProfileIdentifier>
                  <name>PlaceGeomTestV3</name>
                </ProfileIdentifier>
              </identifier>
              <metadata>
                <KeyValuePair>
                  <key>isAbstract</key>
                  <value>false</value>
                </KeyValuePair>
              </metadata>
            </Profile>
          </profile>
        </Requirement>
      </requirement>
    </Schema>
  </schema>
  <schema>
    <Schema>
      <packageName>Schema B</packageName>
      <requirement>
        <Requirement>
          <class>FeatureType1</class>
          <profile>
```

```
          <Profile>
            <identifier>
              <ProfileIdentifier>
                <name>ProfileA</name>
              </ProfileIdentifier>
            </identifier>
          </Profile>
        </profile>
        <profile>
          <Profile>
            <identifier>
              <ProfileIdentifier>
                <name>ProfileB</name>
              </ProfileIdentifier>
            </identifier>
          </Profile>
        </profile>
      </Requirement>
    </requirement>
    <requirement>
      <Requirement>
        <class>FeatureType2</class>
        <profile>
          <Profile>
            <identifier>
              <ProfileIdentifier>
                <name>ProfileA</name>
              </ProfileIdentifier>
            </identifier>
          </Profile>
        </profile>
      </Requirement>
    </requirement>
    <requirement>
      <Requirement>
        <class>FeatureType2</class>
        <property>attribute1</property>
        <profile>
          <Profile>
            <identifier>
              <ProfileIdentifier>
                <name>ProfileA</name>
              </ProfileIdentifier>
            </identifier>
          </Profile>
        </profile>
      </Requirement>
    </requirement>
    <requirement>
      <Requirement>
        <class>FeatureType2</class>
```

```
          <property>attribute2</property>
          <profile>
            <Profile>
              <identifier>
                <ProfileIdentifier>
                  <name>ProfileA</name>
                </ProfileIdentifier>
              </identifier>
            </Profile>
          </profile>
        </Requirement>
      </requirement>
    </Schema>
  </schema>
</ProfileInformation>
```

This example contains profile information for two schemas. It shows that the ProfileLoader can add profile information to multiple schemas contained in the model.

Table 5. ProfileLoader - Parameter(s)

Parameter Name	Type	Required / Optional	Default Value	Description
inputFile	String	Required	-	Location of the file with profile information. This can be a file path (absolute or relative to the working directory) and URL.

For each requirement from the input file the ProfileLoader creates a *profiles* (tagged) value and adds it to the class or property specified by the requirement. By default, all preexisting *profiles* tagged values on this model element will be deleted before the new tagged value is added. If schema elements already have a *profiles* tagged value and the input file does not contain specific profile information for this element, the tagged value will be kept.

The optional rule-trf-profileloader-deleteAllProfilesTaggedValues supports deletion of all *profiles* tagged values in the model before adding new ones.

The default behavior thus supports updating profile information for parts of the schemas that are selected for processing. It does not require profile information to be provided for all (relevant) elements of these schemas.

Table 6. ProfileLoader - Rules

Rule ID	Parameters	Map Entries	Behavior
rule-trf-profileloader-deleteAllProfilesTaggedValues	-	-	If this rule is configured, the ProfileLoader deletes all *profiles* tagged values in the application schemas that are selected for processing before new *profiles* tagged values are generated based upon the content of the input file.

7.3.3. Profiler

This transformation creates a subset of a given model. It is documented in detail on http://shapechange.net/transformations/profiler

In the following we highlight the enhancements specified in Testbed 12, assigned to the three processing phases.

Preprocessing

Before profiling a model, the profiler can perform a number of checks:

- Check that the profile information of a model element is well-formed. If an inconsistent value is found, a warning will be logged. This check is extended as follows:
  - check profile information of profile constraints
  - the check of profile information takes the occurrence of profile metadata into account
- Check that the profile information of a model element is consistent regarding the model, i.e. the profile set of a class contains the profile sets of all its subclasses and properties. This check is extended as follows:
  - check that the profile set of a class also contains the profile sets of its constraints

Processing

During the main processing phase, the Profiler creates the profile (i.e. removes properties and classes that don't belong to the profile).

When the Profiler processes a class or a property, the Profiler first checks if it belongs to the target profiles. If it does not, then the class/property is removed from the model, including any constraints that are defined for it. Otherwise the model element is kept as is, and new/updated behavior is executed: *metadata handling* and *constraint handling*.

Metadata Handling

If profile metadata in the form of key-value-pairs is provided in the profile that the model element (class/property) belongs to, then:

- All keys recognized by the profiler are used to modify the element; the following keys are defined:
  - key: *isAbstract* → abstractness is set according to the value (either "false" or "true", ignoring case); if the value cannot be parsed to a Boolean or if the model element is not a class, a warning is logged and the key-value-pair ignored
  - key: *multiplicity* → the multiplicity is set to the according value (e.g. "0..1"); if the value cannot be parsed to a multiplicity or if the model element is not a property, a warning is logged and the key-value-pair ignored
  - key: *isOrdered* → ordered constraint on property is set according to the value (either "false" or "true", ignoring case); if the value cannot be parsed to a Boolean or if the model element

is not a property, a warning is logged and the key-value-pair ignored

- key: *isUnique* → uniqueness constraint on property is set according to the value (either "false" or "true", ignoring case); if the value cannot be parsed to a Boolean or if the model element is not a property, a warning is logged and the key-value-pair ignored
- key: *isNavigable* → association end navigability is set according to the value (either "false" or "true", ignoring case); if the value cannot be parsed to a Boolean or if the model element is not an association role, a warning is logged and the key-value-pair ignored

- All keys that are not recognized by the Profiler will automatically be added to the model element as tagged value (name = key & value = value). They can then be used in subsequent processing steps (e.g. by the *Flattener* transformation to only apply the homogeneous geometry rule for specific geometry types).

Constraint Handling

1. Any profile constraint is checked to determine if it belongs to the target profiles. If it does not, it is removed. In that respect, a profile constraint is treated like any other model element (currently: class/property) that has profile information attached to it.
2. The Profiler transformation parameter "constraintHandling" specifies additional behavior for handling constraints while the model is actually being profiled/modified. It applies to all constraints (profile constraints and other constraint types):
   - keep (default): all constraints are kept as-is (i.e. all profile constraints that remain from step 1 and all other types of constraints)
   - remove: all constraints are removed
   - removeByPropertyNameInConstraintName: remove a class constraint if its name contains the name of a property (with the suffix “_Type”) that is being removed through profiling. For example, if the attribute “att” is removed from a class during profiling, then any constraint of that class whose name contains “att_Type” is removed by the Profiler.
3. Once profiling of classes and properties is complete, the Profiler revisits the constraints of the remaining classes and properties to see if there are constraints with the same name. If there are, then:
   - if multiple profile constraints have the same name, ShapeChange logs a warning and only keeps one (which is chosen on an arbitrary basis)
   - if multiple non-profile constraints (for example, OCL constraints) have the same name, ShapeChange logs a warning and only keeps one (which is chosen on an arbitrary basis)
   - if (after the previous two checks), both a profile constraint and a non-profile constraint remain, then:
     i. the non-profile constraint is deleted; this supports revision of constraints
     ii. if the profile constraint has no expression it is also deleted; this supports deletion of a constraint in a profile
4. Finally, the profiler transforms profile constraints into constraints of the according types.

NOTE	rule-trf-profiling-processing-explicitProfileSettings does NOT apply to constraints. A profile constraint always has a defined profile identifier. Non-profile constraints simply apply to the class or profile in any profile that this element belongs to.

Postprocessing

A new postprocessing rule has been defined in Testbed 12: *rule-trf-profiling-postprocessing-removeProfilesTaggedValues*. As the name says, it can be used to delete all *profiles* tagged values in the model profile. This can be useful for cleaning up the model for subsequent processing steps where profile information shall not be included, like writing the profile back into an Enterprise Architect repository.

7.4. Constraint Parsing and Validation

A model transformation - especially profiling - can modify the model in such a way that constraints are no longer valid.

In Testbed 12, the postprocessing functionality available for all transformations has been extended to parse and validate constraints.

If a constraint is invalid in the context of the transformed model, then ShapeChange will log a warning (which includes the reason why the constraint is invalid) and convert the constraint into a simple text constraint, so that processing can proceed. The user can then either modify the input model (e.g. constraint definitions), the transformations (e.g. profiling), or the output (if there were only minor issues).

Parsing and validating constraints of a transformed model can help prevent errors.

Validating constraints at the end of a transformation can also be skipped by including *rule-trf-all-postprocess-skip-constraint-validation* in the rules of that transformation. This can be useful if constraints of transformed models are irrelevant for target processing (i.e., the derivation of the final output).

7.5. Implementation

The common constraint parsing and validation functionality has been implemented in Testbed 12. Due to resource limitations, the implementation of the ConstraintLoader, the ProfileLoader, as well as the extensions to the Profiler had to be postponed. They are future work items.

Chapter 8. UML to RDF/OWL/SKOS

8.1. Overview

One of the goals in Testbed 12 was the conversion of the NSG Application Schema (NAS) into an OWL ontology. During the Testbed, a number of requirements were identified for this conversion. It turned out that the recently published ISO 19150-2 does not support all of these requirements. Therefore, additional conversion rules were defined in Testbed 12, together with adaptations and extensions of the rules from ISO 19150-2. These rules are documented in the following section. Some aspects of the ShapeChange implementation of the conversion rules are documented in the Implementation section. The final section of this chapter documents the encoding rule for deriving an ontology from the NAS.

8.2. Conversion Rules

The following subsections describe how the content of an application schema represented using UML as the conceptual schema language can be converted to RDF/SKOS/OWL elements.

For many schema constructs a default conversion rule exists. Specific rules can be used to augment or replace the default rule.

NOTE	A *conversion rule* describes how a specific aspect of the conceptual schema shall be encoded in the target representation (e.g. RDF/OWL). An *encoding rule* consists of a set of conversion rules – as required by a community.

8.2.1. General

Documentation

The documentation of UML packages, classes, properties, and associations includes the following pieces of generic descriptive information, called *descriptors*:

Table 7. Descriptors

Descriptor Name (and ID)	Explanation
Name (name)	The name of the model element (as named in the source UML, i.e. using upper and lower camel case).
Alias (alias)	An alternative, human-readable name for the model element.
Definition (definition)	The normative specification of the model element.
Description (description)	Additional information about the model element.

Descriptor Name (and ID)	Explanation
Example(s) (example)	Example(s) illustrating the model element.
Legal basis (legalBasis)	The legal basis for the model element. NOTE: This descriptor is optional. For some communities, this information is needed in model documentation generated by ShapeChange.
Data capture statement(s) (dataCaptureStatement)	Statement(s) describing how to capture instances of this model element from the real world.
Primary code (primaryCode)	The primary code for this model element. NOTE: The main code for a model element should be assigned to this descriptor. The primary code may be the only one. Optional additional tagged values may be added for other codes.

NOTE The descriptor ID is used in templates of descriptor targets - see next table.

Typically, a community has a preferred way to model and encode this information. For example, one community may want to encode the definition of a model element in a skos:definition property, while another prefers to encode it as part of an rdfs:comment property. Yet another community may want to do both. ShapeChange can support this type of diversity through *descriptor targets* that are part of the configuration. A *descriptor target* specifies how the content of a specific RDF property (that shall be generated while converting a model element) is constructed from descriptors. It uses a template to do so. The *descriptor target* also takes into account that a model element may not have an actual value for each descriptor, and that some descriptors can have multiple values.

In addition to the well-known descriptors (see previous list), additional descriptive information can be incorporated through UML tagged values from the application schema, as explained in the previous table. For example, the "name" tagged value on classes in the NAS could be used to create skos:prefLabel declarations.

The following table documents the structure of a *descriptor target*. An example is also provided below the table. The XML Schema can be found in the Annex on XML Schema Documents.

Table 8. Descriptor Target

Information Item (configured via XML attribute)	Datatype & Structure	Required / Optional	Default Value	Description
appliesTo	one of "ontology", "class", "conceptscheme", "property", "all"	Optional	"all"	Identifies the type of ontology element to which the DescriptorTarget applies. A value of "all" means that the descriptor applies to an ontology, class, conceptscheme, and property.
target	string; the syntax shall follow QNames, with the prefix being equal to the namespace abbreviation of a namespace that is contained in the configuration of the ShapeChange ontology target	Required	*not applicable*	IRI of an RDF property that will be added with the resource representing the model element as subject. The value is determined by the *template* attribute, with the value format being defined by the *format* attribute.

Information Item (configured via XML attribute)	Datatype & Structure	Required / Optional	Default Value	Description
template	string	Required	*not applicable*	Text template where an occurrence of the field "[[*descriptor-ID*]]" is replaced with the value(s) of that descriptor. The IDs of supported descriptors are listed in the table above. An occurrence of the field "[[TV:name]]" is replaced with the value(s) of the UML tagged value with the given name from the input schema. The content of a tagged value can also be split into multiple strings. In that case, use a field "[[TV(*separator*):name]]". The tagged value will be split around matches of the given separator (which is treated as a literal).
format	enum: *string*, *langString*, or *IRI*	Optional	*langString*	Defines the format of the property value: * *langString*: language-tagged string; the configuration parameter "language" (in the ontology target configuration) provides the value of the language tag * *string*: string without language tag * *IRI*: the value is the IRI of a resource
noValueBehavior	enum: *ignore* or *populateOnce*	Optional	*ignore*	Determines the behavior in the case that no value is available for any of the fields contained in the template: * *ignore*: No target property is created. * *populateOnce*: A single target property is created, with the *noValueText* being used for all fields.
noValueText	string	Optional	*the empty string*	If a descriptor used in a template has no value, then this information item provides the text to use instead (e.g. "N/A" or "FIXME").
multiValueBehavior	enum: either *connectInSingleTarget* or *splitToMultipleTargets*	Optional	*connectInSingleTarget*	Specifies how descriptors with multiple values shall be encoded: * *connectInSingleTarget*: Multiple values of a descriptor contained in the template are combined in a single target RDF property value, using the *multiValueConnectorToken* to combine them. * *splitToMultipleTargets*: Multiple values for one or more descriptors result in multiple target RDF properties, one for each value-combination of multi-valued descriptors (resulting in a permutation of the values of each descriptor contained in the template).
multiValueConnectorToken	string	Optional	*a single space character*	If a descriptor used in a template has multiple values, and the *multiValueBehavior* of the descriptor target is set to *connectInSingleTarget*, then the values are concatenated to a single string value using this token as connector between two values.

The following listing gives an example illustrating the use of descriptor targets:

```
<sc:descriptorTargets xmlns:sc="http://www.interactive-
instruments.de/ShapeChange/Configuration/1.1"
  xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sc:DescriptorTarget target="rdfs:label" template="[[alias]]"/>
  <sc:DescriptorTarget target="rdfs:Comment"
    template="-- Definition: [[definition]] -- Description: [[description]]"
noValue="FIXME"/>
  <sc:DescriptorTarget target="skos:prefLabel" template="[[TV:name]]"/>
  <sc:DescriptorTarget target="skos:altLabel" template="[[TV(|):aliasList]]"
multiValueBehavior="splitToMultipleTargets"/>
  <sc:DescriptorTarget target="rdfs:isDefinedBy"
template="[[TV:informationResourceURI]]" format="IRI"/>
  <sc:DescriptorTarget target="ex:example" template="[[example]]" noValue="n/a"
multiValueConnectorToken="\r\n\r\n"/>
</sc:descriptorTargets>
```

NOTE

- Descriptors can be converted to RDF using a specific language identified via the parameter "language" (in the ontology target configuration). A future work item provides further details.
- Additional annotations (e.g. rdfs:seeAlso and owl:deprecated) for which no information is contained in the application schema - in descriptors or tagged values - need to either be set manually or suitable UML tagged values added to the application schema elements and then realize using descriptor targets.
- The ISO 19150-2 documentation requirements - for example setting rdfs:label and rdfs:isDefinedBy - can be realized using descriptor targets.
- Descriptor targets are processed when encoding UML packages, classes, and properties. They do not apply to constraints.

8.2.2. Package

By default, an ontology with a unique RDF namespace is created for each package that belongs to a schema that is selected for processing by ShapeChange. This matches the behavior described in ISO 19150-2 clause 6.

NOTE As described in the overview, the ShapeChange configuration allows a user to select a specific subset of all schemas contained in the input model. Transformations and targets will only be executed for *selected schemas*.

If *rule-owl-pkg-singleOntologyPerSchema* is enabled, then one ontology namespace will be created for the selected schema and its child packages.

NOTE The content of a UML package, i.e. UML classes and their properties, will be converted to OWL depending upon the conversion rules that have been configured.

Name and Namespace

The *ontology name* is determined by ShapeChange according to the following rules:

- If *rule-owl-pkg-ontologyName-iso191502* is enabled, then the ontology name is the normalized package name appended to a base URI (with “/” as separator).
- If *rule-owl-pkg-ontologyName-byTaggedValue* is included in the encoding rule and a tagged value - identified by the configuration parameter *ontologyNameTaggedValue* (default value: ontologyName) - is set for the package, then its value is used.
  - This option is useful if ontologies shall be derived from multiple schemas, and each shall have a specific ontology name that is defined in the UML model.
- If *rule-owl-pkg-singleOntologyPerSchema* and *rule-owl-pkg-ontologyName-code* are both in effect, the ontology name is constructed as follows: a code value is appended to a base URI (see next NOTE for further details) with "/" as separator.

  The code value is determined by first looking at the configuration parameter *ontologyNameCode*. If it exists, its value is used. Otherwise, the abbreviation defined for an application schema package - via the tagged value *xmlns* on the application schema package - is used for constructing the ontology name. If neither the configuration parameter nor the tagged value exist, or they do not contain a non-empty string, then "FIXME" will be used as the code value.

  The name of the file in which the ontology is stored will then also be constructed using the code value – instead of the package name (which would be normalized according to ISO 19150-2).
  - This option is useful if a community uses specific codes to identify its schemas.
- If *rule-owl-pkg-ontologyName-withPath* is in effect, then the umlPackageName (that is appended to a base URI with "/" as separator) is constructed using the path of the package to the upmost owner that is in the same target namespace - using a combination of "/" and normalized package names for all parent packages in the same target namespace.
  - This option is useful if child packages of a schema shall be converted to ontologies with different names, and the package names are suitable to construct the ontology name.

NOTE	The base URI is determined as follows: if the configuration parameter 'URIbase' is set, then its value is used - otherwise the targetNamespace of the package provides the value of the base URI.

If *rule-owl-pkg-ontologyName-appendVersion* is enabled, and the 'version' tagged value of the application schema package contains a non-empty string, then the version is appended to the ontology name with separator "/".

NOTE	Although including a version number in the ontology name is not common practice, it can be useful to better enforce use of a particular version of an ontology within a community.

The *RDF namespace* of an ontology - also known as the *ontology IRI* - is constructed by appending a

specific character to the *ontology name*. ISO 19150-2 specifies that character as '#':

```
rdfNamespace = ontologyName "#"
```

This can be changed - for example to '/' - using the ShapeChange configuration parameter *rdfNamespaceSeparator*.

NOTE	Using '/' as separator allows access to individual resources (e.g. classes, properties) within an ontology. This can be useful for large ontologies.

Rule execution priority and dependencies

The conversion rules to determine the ontology name are executed with the following priority:

1. *rule-owl-pkg-ontologyName-byTaggedValue*
2. *rule-owl-pkg-ontologyName-code* (only in combination with *rule-owl-pkg-singleOntologyPerSchema*)
3. *rule-owl-pkg-ontologyName-withPath*
4. *rule-owl-pkg-ontologyName-iso191502*

If none of these rules is enabled, ShapeChange will log a warning and use *rule-owl-pkg-ontologyName-iso191502* as fallback.

Appending a version to the ontology name is an additional processing step. Therefore, *rule-owl-pkg-ontologyName-appendVersion* is independent of the other rules.

Version Information

If a package that is converted into an ontology has version information and *rule-owl-pkg-versionInfo* is enabled, then the information is encoded in an owl:versionInfo element.

NOTE	ShapeChange looks up the version of a package either directly in the configuration (more specifically: in PackageInfo elements) or in the tagged value "version" of the package. If no version information is found there, the version of the parent package is checked. If the parent package has version information, it is used for the child package. Because of this, <<leaf>> packages of an application schema usually have the same version as the application schema. This can be useful if ontologies are created for all packages within a given application schema.

If *rule-owl-pkg-versionIRI* is included and a package that is converted into an ontology has version information, then the versionIRI of the ontology is constructed as follows:

```
ontologyName "/" version
```

If *rule-owl-pkg-versionIRI* and *rule-owl-pkg-ontologyName-appendVersion* are both enabled, and the package has version information, then the version would be duplicated in the versionIRI. To avoid

this, enable *rule-owl-pkg-versionIRI-avoid-duplicate-version*.

NOTE	No rules are available to set additional annotations related to versioning of ontologies (owl:priorVersion, owl:backwardCompatibleWith, owl:incompatibleWith). If necessary, they need to be set manually.

Package Documentation

In addition to the general documentation rules, if *rule-owl-pkg-dctSourceTitle* is enabled, then the conversion of a UML package into an ontology also adds information about the title of the specification or standard that is the source of the ontology definitions, using a dct:source element as described by ISO 19150-2.

The value of this element is computed as follows:

- If the configuration parameter "sourceTaggedValueName" is set and the package has this tagged value, its value is used.
- Otherwise, if the configuration parameter "source" is set, then its value is used.
- Otherwise the value "FIXME" is used (obviously, this requires manual adjustment after the ontology has been created by ShapeChange).

Imports

The import of an external ontology is specified using an owl:imports declaration.

An owl:imports is required in the following cases:

- The application schema uses one or more types from an external schema, for example in inheritance relationships or as value types. In this case, the ontology that is derived from the application schema must import the RDF/OWL implementation of the external schema.
  More specifically, the RDF/OWL implementation of the types from the external schema must be imported. This implementation is not necessarily a single ontology. As discussed before, the packages of a schema (in this case: the external schema) may have been converted to multiple ontologies. Also, ShapeChange supports mappings of individual types and properties from the UML model to any RDF/OWL representation.
- The conversion of the application schema to an ontology can add additional relationships to RDF/OWL classes and properties (for example, using the rdfs:subClassOf predicate). These relationships are not necessarily modelled in UML. However, they need to be represented in the resulting ontology. That requires importing the ontologies that define these RDF/OWL classes and properties.

If *rule-owl-pkg-importISO191502Base* is included, then each ontology imports the base ontology defined by ISO 19150-2 with IRI http://def.isotc211.org/iso19150-2/2012/base#.

8.2.3. Class

General

Unless stated otherwise for specific types of classes (e.g. *enumerations*), a UML class is converted to an owl:Class.

However, in general, if an RdfTypeMapEntry in the ShapeChange configuration provides a mapping for a UML type from the processed schema, then the UML type is not itself encoded. Instead, whenever the UML type is used in the schema, the target RDFS or OWL class specified by the map entry is used in the RDF implementation.

For example, the following configuration fragment specifies that:

- UML type "Type100" in schema "MySchema" is implemented by class http://example.org/1#Z.
- UML type "Type101" in any schema is implemented by datatype http://example.org/1#Y

```
<sc:rdfMapEntries>
  <sc:RdfTypeMapEntry type="Type100" schema="MySchema" target="ex1:Z"/>
  <sc:RdfTypeMapEntry type="Type101" target="ex1:Y" targetType="datatype"/>
</sc:rdfMapEntries>
<sc:namespaces>
  <sc:Namespace nsabr="ex1" ns="http://example.org/1#"/>
</sc:namespaces>
```

The XML Schema definition of rdfMapEntries (including documentation of attributes) is provided in Annex B.

NOTE Provision of the "targetType" in an RdfTypeMapEntry is necessary to differentiate if the target is an RDFS class or a datatype. UML properties with the type stated in the map entry as value type can then be encoded accordingly. The default value for "targetType" is "class".

NOTE The configuration may contain multiple RdfTypeMapEntry elements (RTMEs) for a specific type (T), each with a different 'schema'. The look-up of the RdfTypeMapEntry that applies to T is performed as follows:

- If one RdfTypeMapEntry from RTME has a schema that matches the schema of T then that map entry is chosen (because it is specific for T).
- Otherwise, if one RdfTypeMapEntry from RTME does not define any schema, then it is chosen (because it is a generic mapping for T).
- Otherwise, none of the elements in RTME applies to T.

Class Name

Following ISO 19150-2, the OWL class name is a combination of the RDF namespace of the ontology and the UML class name:

```
className = rdfNamespace umlClassName
```

The UML class name is given in upper camel case. Punctuation characters other than dash and underscore are replaced by underscore characters. Space characters are removed.

NOTE	The rule for constructing the name is the same for an OWL class and an OWL datatype.

Abstract class

If *rule-owl-cls-iso191502IsAbstract* is enabled, then the owl:Class representation of an abstract class from the schema receives the annotation property iso19150-2:isAbstract with value set to *true*.

Generalization/Inheritance

If *rule-owl-cls-generalization* is enabled, then a generalization relationship between a class in the schema and another class is implemented as an rdfs:subClassOf declaration.

Uniqueness / Disjointness

As described in the section on targeted solutions, ISO 19150-2 does not address the representation of uniqueness/disjointness of subtypes in inheritance trees.

If *rule-owl-cls-disjoint-classes* is enabled, then an OWL disjoint classes axiom is created for each case where a supertype has more than one subtype. The axiom contains a collection of the direct subtypes of the supertype, thus ensuring that they are mutually disjoint.

Consider the following example:

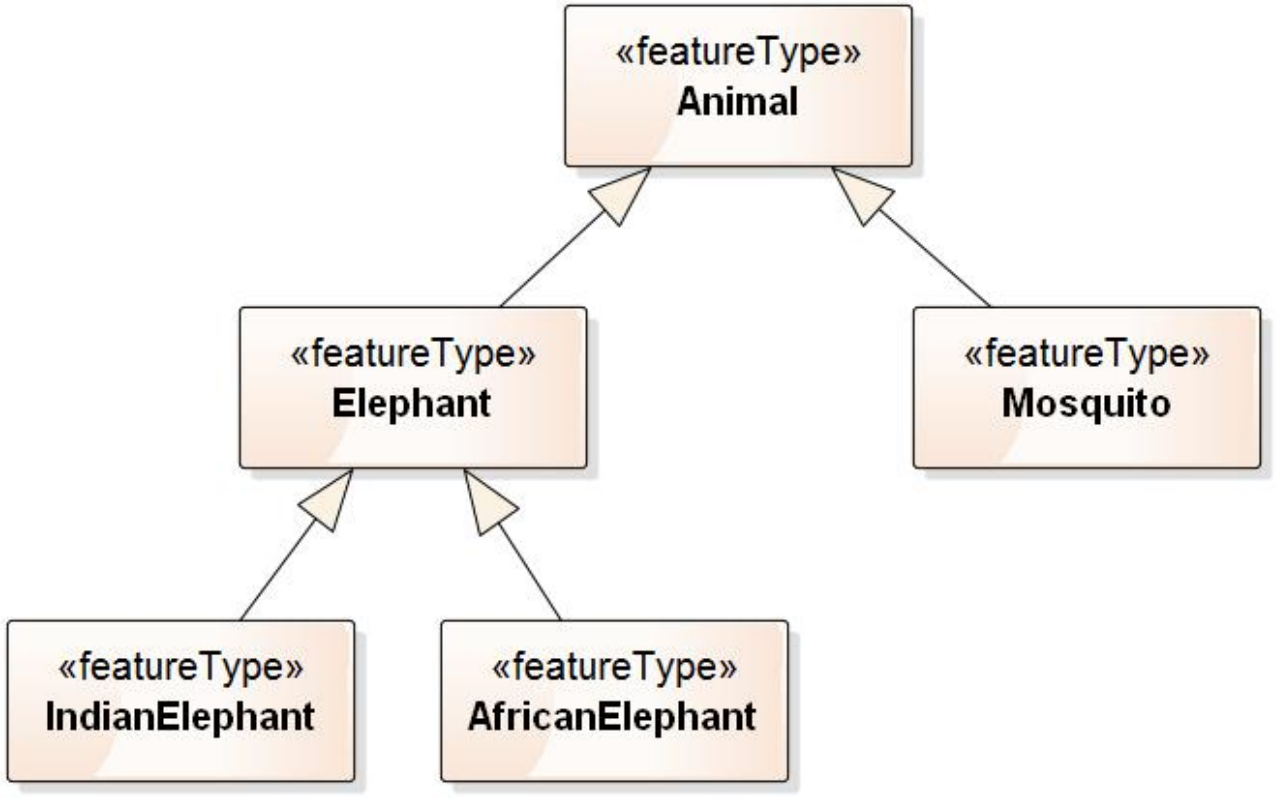

Figure 5. Inheritance example to illustrate conversion of uniqueness

The conversion rule would create two OWL DisjointClasses axioms to represent this inheritance tree:

- DisjointClasses(ex:Elephant ex:Mosquito)
- DisjointClasses(ex:IndianElephant ex:AfricanElephant)

The first axiom ensures that an animal can either be an elephant or a mosquito (this includes subclasses - so an Indian elephant cannot be a mosquito) but not both. Likewise, the second axiom ensures that an elephant can either be an Indian elephant or an African elephant, but not both.

Multiple inheritance is covered as well - consider the following figure.

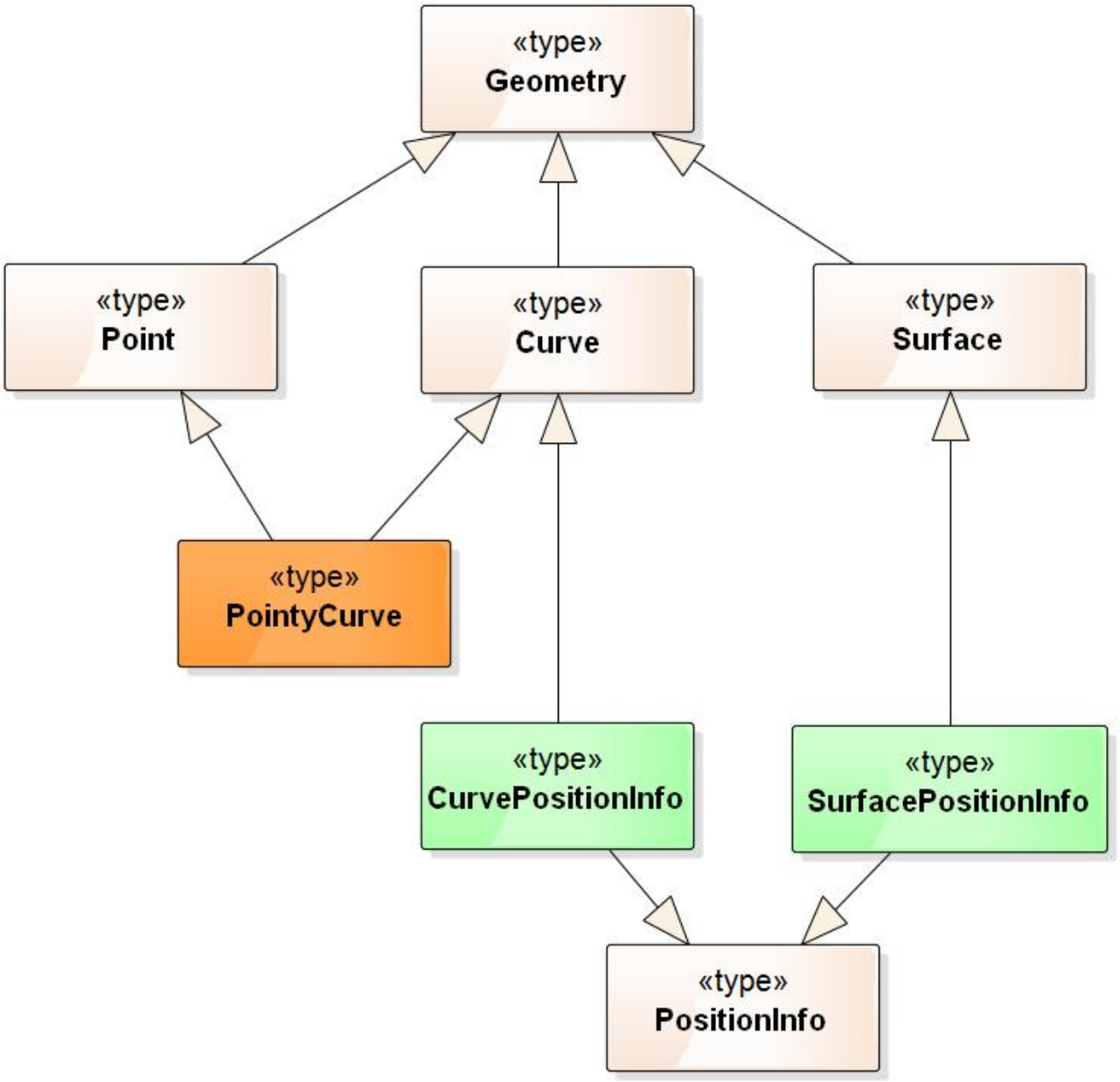

Figure 6. uniqueness / disjointness - multiple inheritance example

As long as the supertypes (including all their supertypes up to the top of the inheritance tree) of a class are not disjoint, such a class can exist. In the example, *Point*, *Curve*, and *Surface* are disjoint according to *rule-owl-cls-disjoint-classes* applied to the supertype *Geometry*. By the same rule, *CurvePositionInfo* and *SurfacePositionInfo* are disjoint subclasses of *PositionInfo*. A *PointyCurve* individual cannot exist because *Point* and *Curve* are disjoint. However, individuals belonging to *CurvePositionInfo* can exist, because its superclasses (*Curve* and *PositionInfo*) are not disjoint. Similarly, individuals belonging to *SurfacePositionInfo* may also exist.

Custom subClassOf Mappings

If required by a specific encoding rule, rdfs:subClassOf properties can be added to particular classes and types of classes (identified by stereotypes). The needed conversion information is kept separate from the model. It is provided in the ShapeChange configuration, using the configuration elements *StereotypeConversionParameter* and *TypeConversionParameter*.

For example, the following configuration fragment specifies that:

- All <<featureType>> classes shall be rdfs:subClassOf GeoSPARQL "Feature".
- The UML type "Feature1" from "Schema 1" shall be rdfs:subClassOf the class http://example.org/1#C.

```
  <sc:rdfConversionParameters>
    <sc:StereotypeConversionParameter wellknown="FeatureType"
subClassOf="geosparql:Feature"/>
    <sc:TypeConversionParameter type="Feature1" schema="Schema 1" subClassOf="ex1:C"/>
  </sc:rdfConversionParameters>
  <sc:namespaces>
    <sc:Namespace nsabr="geosparql" ns="http://www.opengis.net/ont/geosparql#"/>
    <sc:Namespace nsabr="ex1" ns="http://example.org/1#"/>
  </sc:namespaces>
```

The XML Schema definition of rdfConversionParameters (including documentation of attributes) is provided in Annex B.

NOTE

- If both an RDF map entry and an RDF conversion parameter apply to the same UML type (or UML property), then the RDF map entry has higher priority (and the RDF conversion parameter will be ignored).
- The "schema" attribute in a *TypeConversionParameter* element can be used to identify the application schema to which the "type" belongs.

The configuration may contain multiple TypeConversionParameter elements (TCP) for a specific type (T), each with a different 'schema'. The look-up of the TypeConversionParameter that applies to T is performed as follows:

- If one TypeConversionParameter from TCP has a schema that matches the schema of T, then that conversion parameter is chosen (because it is specific for T).
- Otherwise, if one TypeConversionParameter from TCP does not define any schema, then it is chosen (because it is a generic mapping for T).
- Otherwise, none of the elements in TCP applies to T.

Feature Types

If *rule-owl-cls-encode-featuretypes* is enabled, feature types will be converted to OWL classes.

Object Types

ShapeChange treats UML types without stereotype as plain objects with identity, called *object types*. UML types with stereotype <<type>> are usually also treated this way.

If *rule-owl-cls-encode-objecttypes* is enabled, object types will be converted to OWL classes.

NOTE

Omitting this encoding rule can be useful when creating a taxonomy of particular UML types.

Mixin Types

ShapeChange supports the notion of *mixin type* (for further details, see http://shapechange.net/targets/xsd/extensions/mixin/). They are primarily used by the XML Schema target. However, if that target is contained in the ShapeChange configuration, it has implications on how UML types are loaded. In this case, it may lead to UML types being loaded as *mixin types*. A UML type is loaded as a mixin type if:

- rule-xsd-cls-mixin-classes is contained in the XSD encoding rule and:
  - the tagged value gmlMixin is set to true, or
  - The type has the stereotype <<type>>, is abstract, and the tagged value gmlMixin is not set to false.

For the ontology conversion rules, a *mixin type* is usually treated as any other object type. If UML types are loaded by ShapeChange as mixin types, include *rule-owl-cls-encode-mixintypes* to convert them to OWL classes.

NOTE	Omitting this encoding rule can be useful when creating a taxonomy of particular UML types.

Data Types

If *rule-owl-cls-encode-datatypes* is enabled, data types will be converted to OWL classes.

NOTE	Omitting this encoding rule can be useful when creating a taxonomy of particular UML types.

Basic Types

ShapeChange supports the notion of *basic type* (for further details, see http://shapechange.net/targets/xsd/extensions/#rule-xsd-cls-basictype). They are primarily used by the XML Schema target. However, if that target is contained in the ShapeChange configuration, it has implications on how UML types are loaded. In this case, it may lead to UML types being loaded as *basic types*. A UML type is loaded as a basic type if:

- rule-xsd-cls-basictype is contained in the XSD encoding rule, and:
  - the UML type has the stereotype <<BasicType>>, or
  - the UML type is an object type, mixin type, or data type and has a supertype that is identified as a basic type through an XSD map entry (which does not have its *xmlTypeType* and *xmlTypeContent* attributes both set to 'complex' - either explicitly or by using the default values, which is 'complex' for both attributes).

For the ontology conversion rules, a *basic type* is usually treated as any other type. If UML types are loaded by ShapeChange as basic types, include *rule-owl-cls-encode-basictypes* to convert them to OWL classes.

NOTE	Omitting this encoding rule can be useful when creating a taxonomy of particular UML types.

Union

As outlined in the section on targeted solutions, the conversion of a <<union>> type as specified by ISO 19150-2 is insufficient.

In the following, we define a universal rule to convert unions that solves these issues. At the same time, we explain why the universal rule may not always work well with common semantic web software. Therefore, we propose two alternative rules, one being specific to how unions are used in the NAS modeling pattern for XxxMeta datatypes.

Universal Approach

This approach is based on a logical combination of unqualified cardinality constraints. The resulting class expression ensures that values are given for only one of the properties in the union.

More specifically, the class expression is created as follows:

- For each property P in the <<union>> type:
  - Create a set of unqualified cardinality restrictions, to express the multiplicity of P (for example by restricting the minimum cardinality of P to 1), and that the cardinality of each of the other properties is restricted to 0.
    - NOTE: If multiplicity is not encoded (neither *rule-owl-prop-multiplicityAsQualifiedCardinalityRestriction* nor *rule-owl-prop-multiplicityAsUnqualifiedCardinalityRestriction* is part of the encoding rule), then the minimum cardinality of P would explicitly be set to zero, so that the combination of cardinality restrictions covers the other properties as well as P.
  - Combine the set of cardinality restrictions within an *intersection class expression*. The meaning of this expression is that if a value is given for P, then no values may be provided for the other new properties.
- Combine the set of property-specific intersection class expressions within a single *union class expression*. This essentially represents the meaning of the <<union>> type: if a value is given for one of the options, then there must not be values for the other options.

The class expression is added to the OWL class representation of the <<union>> type.

Each <<union>> type property is converted to an OWL property according to the general rules defined for properties. This ensures that the value type of the property can be represented properly.

Consider the following example:

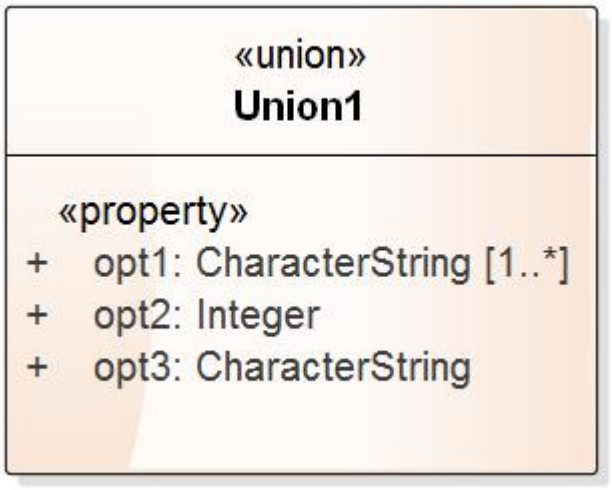

Figure 7. Example of a <<union>>

In pseudocode, the class expression that declares the union semantics can be described as follows, where an integer indicates the number of values that exist for a given property:

```
   (#opt1 >= 1 AND #opt2 = 0 AND #opt3 = 0)
OR (#opt1 = 0 AND #opt2 = 1 AND #opt3 = 0)
OR (#opt1 = 0 AND #opt2 = 0 AND #opt3 = 1)
```

The class expression ensures that values exist only for one of the union properties, with the number of values constrained as defined by the multiplicity of that property in the UML model.

The OWL encoding of Union1 would be (namespaces omitted for brevity):

```
:opt1 rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:string .

:opt2 rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:integer .

:opt3 rdf:type owl:DatatypeProperty ;
  rdfs:range xsd:string .

:Union1 rdf:type owl:Class ;

  rdfs:subClassOf [
    rdf:type owl:Class ;
    owl:unionOf (
      [ owl:intersectionOf (
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt1 ;
          owl:minCardinality "1"^^xsd:nonNegativeInteger]
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt2 ;
          owl:cardinality "0"^^xsd:nonNegativeInteger]
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt3 ;
          owl:cardinality "0"^^xsd:nonNegativeInteger]) ;
       rdf:type owl:Class]
      [ owl:intersectionOf (
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt1 ;
          owl:cardinality "0"^^xsd:nonNegativeInteger]
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt2 ;
          owl:cardinality "1"^^xsd:nonNegativeInteger]
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt3 ;
          owl:cardinality "0"^^xsd:nonNegativeInteger]) ;
       rdf:type owl:Class]
     [ owl:intersectionOf (
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt1 ;
          owl:cardinality "0"^^xsd:nonNegativeInteger]
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt2 ;
          owl:cardinality "0"^^xsd:nonNegativeInteger]
        [ rdf:type owl:Restriction ;
          owl:onProperty :opt3 ;
          owl:cardinality "1"^^xsd:nonNegativeInteger]) ;
       rdf:type owl:Class]
    )
  ] .
```

The identifier for this conversion rule is: *rule-owl-cls-union*.

NOTE

As described in the section on multiplicity an encoding rule may omit conversion rules to handle property multiplicity. In that case, the multiplicity defined by the UML model is not specified by cardinality restrictions in the resulting ontology. Nevertheless, to model the semantics of a union in OWL, a class expression as described above would still be used - just with owl:minCardinality of 0 as the cardinality restriction for each union property. In pseudocode:

```
   (#opt1 >= 0 AND #opt2 = 0 AND #opt3 = 0)
OR (#opt1 = 0 AND #opt2 >= 0 AND #opt3 = 0)
OR (#opt1 = 0 AND #opt2 = 0 AND #opt3 >= 0)
```

If multiplicity is encoded, then the class expression that defines the union semantics already defines the multiplicity. Therefore, additional cardinality restrictions for the union options are not encoded. Actually, if they were encoded, it would often lead to a class definition that cannot be satisfied: if two options have minimum multiplicity of 1, then the cardinality restrictions to represent multiplicity would require that an instance of a union had values for both properties, which contradicts the class expression that represents the union semantics - where only one of the properties can have a value.

However, we think that this universal approach might not work well in actual applications. Because the union is represented as a class, individuals of that union type encoded in RDF would contain the evaluated union property. A union individual would be an intermediary class between the value of the property and the individual described by that value. Usually, RDF applications can find the actual value of a property - for an individual of a "normal" class - directly, i.e. without having to take an intermediate step (in this case through the "union" individual). It is not clear how well semantic web software would be able to handle "union" classes as described in this universal approach.

Alternative: Flattening the Union

In this approach, the union class itself is not converted to an OWL class. Instead, each UML property - typically a UML attribute - that has a union as value type (a property referred to as "A" in the following definitions) is converted by replacing it with new properties, one for each property option (in the following referenced as "B") of the union.

The following paragraphs explain how the name, descriptors (e.g. definition and description), multiplicity, and value type of the new properties are determined.

The name of a new property is constructed as follows: <name property A> <union_separator> <name property B>.

The *union_separator* is defined in the configuration parameter "separatorForPropertyFromUnion".

For example, where the *union_separator* is set to "_by_", the resulting property names from flattening a union with two properties are "vulnerabilityToPollution_by_singleQuantity" and

"vulnerabilityToPollution_by_quantityInterval".

In some cases, the name of the union property B already contains the name of property A. For example: "areaOfResponsibility" (name of property A) and "areaOfResponsibilityByPolygon" (name of the union property B). Concatenation of names in such a case would lead to duplication (in the example: areaOfResponsibility_by_areaOfResponsibilityByPolygon). This can be corrected through a manual review of the resulting ontology.

NOTE	An approach for automatically detecting and avoiding such duplication may become available in the future.

The descriptors of properties A and B are merged:

- Definition, description, primary code, and legal basis are concatenated, using a single space as separator.
- The data capture statements are merged into a single list.
- The examples are merged into a single list.

This automatic combination may not always be ideal. Therefore, a manual review of the resulting ontology should be performed.

The handling of tagged values also needs to be defined. The default behavior - which can be modified in the future through additional rules - is to ignore the tagged values from property A and to use the tagged values from property B for the new property. The *sequenceNumber* tagged value is a special case, because it will need to be adjusted so that the new property is correctly positioned within its class, without any duplication of sequence numbers.

Each new property receives the value type of the union property as its value type.

The multiplicity of the new property is defined as follows:

- lower bound: If maximum multiplicity of A is 1 then the lower bound is the product of the minimum multiplicities of property A and B; otherwise it is 0.
  - NOTE: If the maximum multiplicity of A is 1, then a class expression as defined in the universal approach can be used to represent the union semantics; in that case, the actual minimum multiplicity of A is of interest (so should not always be set to 0).
- upper bound: The product of the maximum multiplicities of property A and B - or unbounded if one of the maximum multiplicities is unbounded.

Cardinality restrictions can be used to express the resulting multiplicity for each new property.

NOTE	If the lower bound of the multiplicities of properties A or B is 0, then the cardinality restriction for the new property should use owl:minCardinality=0. This supports the situation in which the new property does not have a value at all. In most cases, property B - so a <<union>> option - will have a lower bound of 1. Property A, on the other hand, can have a lower bound of 0.

As described in the universal approach, an OWL class expression can be used to express that an

individual can only have values for one of the new properties (or none at all), thus specifying the union semantics. Such a class expression is only created if the maximum multiplicity of property A is one. In this case, the restriction to one of the new properties is correct.

NOTE If the maximum multiplicity of A is greater than one, then it can have values from more than one option. In that case, it would not be correct to add a class expression that restricts values to only one of the new properties.

The rule to transform the model in this way has the identifier *rule-trf-prop-flatten-types*. The following listing provides an example of a ShapeChange configuration element, with parameters and this rule, with which this transformation can be achieved. For further details about the transformation rule and its parameters, see http://shapechange.net/transformations/flattener/#rule-trf-prop-flatten-types

```
    <Transformer
class="de.interactive_instruments.ShapeChange.Transformation.Flattening.Flattener"
      input="A" id="B" mode="enabled">
      <parameters>
        <ProcessParameter name="includeUnionIdentifierTaggedValue" value="true"/>
        <ProcessParameter name="mergeDescriptors" value="true"/>
        <ProcessParameter name="separatorForPropertyFromUnion" value="_"/>
        <ProcessParameter name="flattenObjectTypes" value="false"/>
        <ProcessParameter name="flattenDataTypesExcludeRegex" value=".*"/>
        <ProcessParameter name="setMinCardinalityToZeroWhenMergingUnion" value=
"false"/>
      </parameters>
      <rules>
        <ProcessRuleSet name="flattener">
          <rule name="rule-trf-prop-flatten-types"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
```

During the transformation, ShapeChange sets tagged values (tag name is: 'SC_UNION_SET') to keep track of which properties belong to which union. When converting the model to an ontology, the conversion rule *rule-owl-cls-unionSets* can make use of this information to create class expressions that represent the union semantics.

Alternative: replace property with union as value type with union options

In special cases, a simpler flattening approach may be applied. If only a single property (A) of a non-union type (e.g. a datatype) has a specific union as value type, and if that property has maximum multiplicity 1, then merging property names would not be necessary. In that case, the union properties can simply replace property A without merging. This would not produce ambiguity in the non-union type. Only the sequenceNumbers will need to be adjusted, so that the union properties are correctly positioned within their new class.

NOTE The mentioned preconditions can automatically be checked by ShapeChange.

In the NAS, for example, unions represent choices between an actual "value" for a property and a "reason" why the value is not provided. The union is always used by a single property "valueOrReason" or "valuesOrReason" of an XxxMeta datatype. The following figure provides an example.

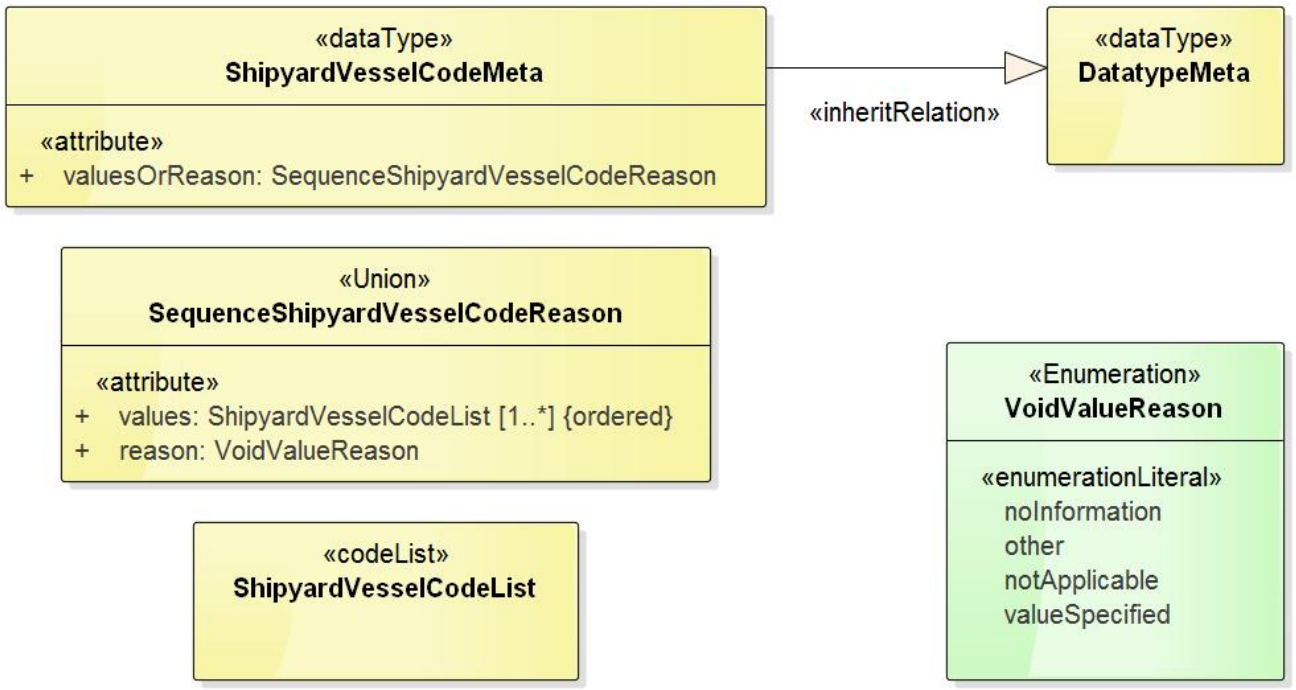

Figure 8. Example of <<union>> usage within the NAS

Both "value" and "reason" can directly replace "valuesOrReason" within the ShipyardVesselCodeMeta datatype.

NOTE

The OWL property name is constructed using the XxxMeta class name, because both "value" and "reason" are not global properties. Encoding of "reason" as a global property would make sense in this specific case because the semantics are the same for each XxxReason <<union>>. This would work for the conversion of NAS unions, because an XxxMeta datatype only has a single property with union value type. If multiple of these unions were used in the XxxMeta datatype and "reason" was a global property, then it would not be possible to differentiate for which particular property a "reason" is provided.

The rule to transform the model in this way has the identifier *rule-trf-cls-replace-with-union-properties*. The following listing provides an example of a ShapeChange configuration element, with parameters and this rule, with which this transformation can be achieved. For further details about the transformation rule and its parameters, see http://shapechange.net/transformations/flattener/#rule-trf-cls-replace-with-union-properties

```
    <Transformer
class="de.interactive_instruments.ShapeChange.Transformation.Flattening.Flattener"
      input="A" id="B" mode="enabled">
      <parameters>
        <ProcessParameter name="includeUnionIdentifierTaggedValue" value="true"/>
      </parameters>
      <rules>
        <ProcessRuleSet name="flattener">
          <rule name="rule-trf-cls-replace-with-union-properties"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
```

As described before, ShapeChange sets tagged values (tag name is: 'SC_UNION_SET') during the transformation to keep track of which properties belong to which union. When converting the model to an ontology, the conversion rule *rule-owl-cls-unionSets* can make use of this information to create class expressions that represent the union semantics.

Rule Execution Priority and Dependencies

The conversion rules for unions - *rule-owl-cls-union* and *rule-owl-cls-unionSets* - are independent of each other.

If none of these rules is enabled, unions and union semantics will not be encoded. The range of properties with a union as value type will then be set to the value of the ShapeChange configuration parameter *defaultTypeImplementation* (default is owl:Class).

Enumeration

ISO 19150-2

The conversion of enumerations as defined by ISO 19150-2 is covered by *rule-owl-cls-iso191502Enumeration*. The enumeration corresponds to an RDFS datatype that specifies the restricted list of literals using an owl:oneOf declaration.

Enumeration as Code List

An alternative conversion rule is also available: *rule-owl-cls-enumerationAsCodelist*. It takes into account that the conceptual difference between enumerations and code lists is rather small. An enumeration is considered to be a complete list of codes, while a code list is extensible. Both contain lists of acceptable values to be used with properties, typically as strings of natural language or mnemonic abbreviations. Optionally (and preferably) both kinds of elements are provided with definitions.
Due to their dynamic nature, code lists are now often encoded (e.g. using SKOS) and managed separately from the application schema. SKOS representation can be applied to enumerations as well. If *rule-owl-cls-enumerationAsCodelist* is enabled, then the conversion to RDF treats enumerations as code lists.

NOTE	That means that tagged values *vocabulary* and *codeList* can also be relevant on enumerations. If *rule-owl-cls-enumerationAsCodelist* and *rule-owl-cls-codelist-external* are enabled and an enumeration has the tagged value *vocabulary* or *codeList*, then the enumeration will be treated as externally managed. It will not be converted to an OWL class and SKOS individuals; instead, the OWL representation of UML properties that have the enumeration as value type will reference the external representation of the enumeration (see section on code lists for further details).

Treating enumerations as code lists may also be desirable for the following reasons:

- The representation of enumerations as string values (within an owl:oneOf declaration as specified by ISO 19150-2) does not allow for providing definitions of the enumerants within the ontology.
- The approach in ISO 19150-2 makes the representation of enumerations very different from the representation of code lists, even though conceptually the two are similar except for enumerations being supposed to be a complete, closed coverage of the value space.
- Applications that are designed to utilize controlled vocabularies in SKOS - for example for tagging and indexing - cannot make use of enumerations represented as defined by ISO 19150-2.
- In the RDF/XML serialization of owl:oneOf, the rdf:List structure, with its LISP-like nesting, is unwieldy.

Rule Execution Priority and Dependencies

The conversion rules for enumerations are mutually exclusive. If both rules are enabled, ShapeChange will log a warning and continue processing as if only *rule-owl-cls-iso191502Enumeration* was enabled. If none of these rules is enabled, enumerations will not be encoded. The range of properties with an enumeration as value type will then be set to the value of the ShapeChange configuration parameter *defaultTypeImplementation* (default is owl:Class).

Code Lists

<<codeList>> UML classes can be converted in a number of ways:

- They can be treated as stubs that simply provide a reference to an externally managed resource that represents the codelist.
- They can be converted as specified by ISO 19150-2.

External References

Communities often already manage their code lists outside of the application schema, SKOS being a prominent encoding. In such a case, <<codeList>> UML classes are contained in the application schema without declaring any specific code values. These classes serve as stubs. They can be used as value types for properties. In addition, they can - and should - provide links to the resource that represents the code list using a tagged value.

rule-owl-cls-codelist-external supports such a scenario. If this conversion rule is enabled, then a code list that has the *vocabulary* or *codeList* tagged value is not represented as part of the OWL

ontology derived from the application schema.

In this case, the IRI contained in the tagged value links to a resource that represents the code list. The resource can be a simple web page. It can also be a registry entry, with the registry providing multiple formats of the resource. For example, an HTML page is presented for human readers, while an RDF/OWL/SKOS representation is made available for automated processing. HTTP "Accept" headers can be used to identify the desired format.

However, the IRI often does not directly identify an rdfs:Class (since, as explained before, it may link to different types of resources that represent the code list). A specific range to be used by RDF/OWL properties that have the code list as value is therefore missing.

An RdfTypeMapEntry can be used to define the mapping between a <<codeList>> UML class and an RDFS/OWL class - see the general section on conversion of classes for further details.

If no specific range is available to represent the value type of a UML property, then a globally configured range is used. It can be configured using the parameter *defaultTypeImplementation* (default is owl:Class).

ISO 19150-2

If *rule-owl-cls-codelist-19150-2* is enabled then <<codeList>> UML classes are converted as defined by ISO 19150-2 (section 6.7.2) - except for the SKOS collection (for which a separate conversion rule is defined).

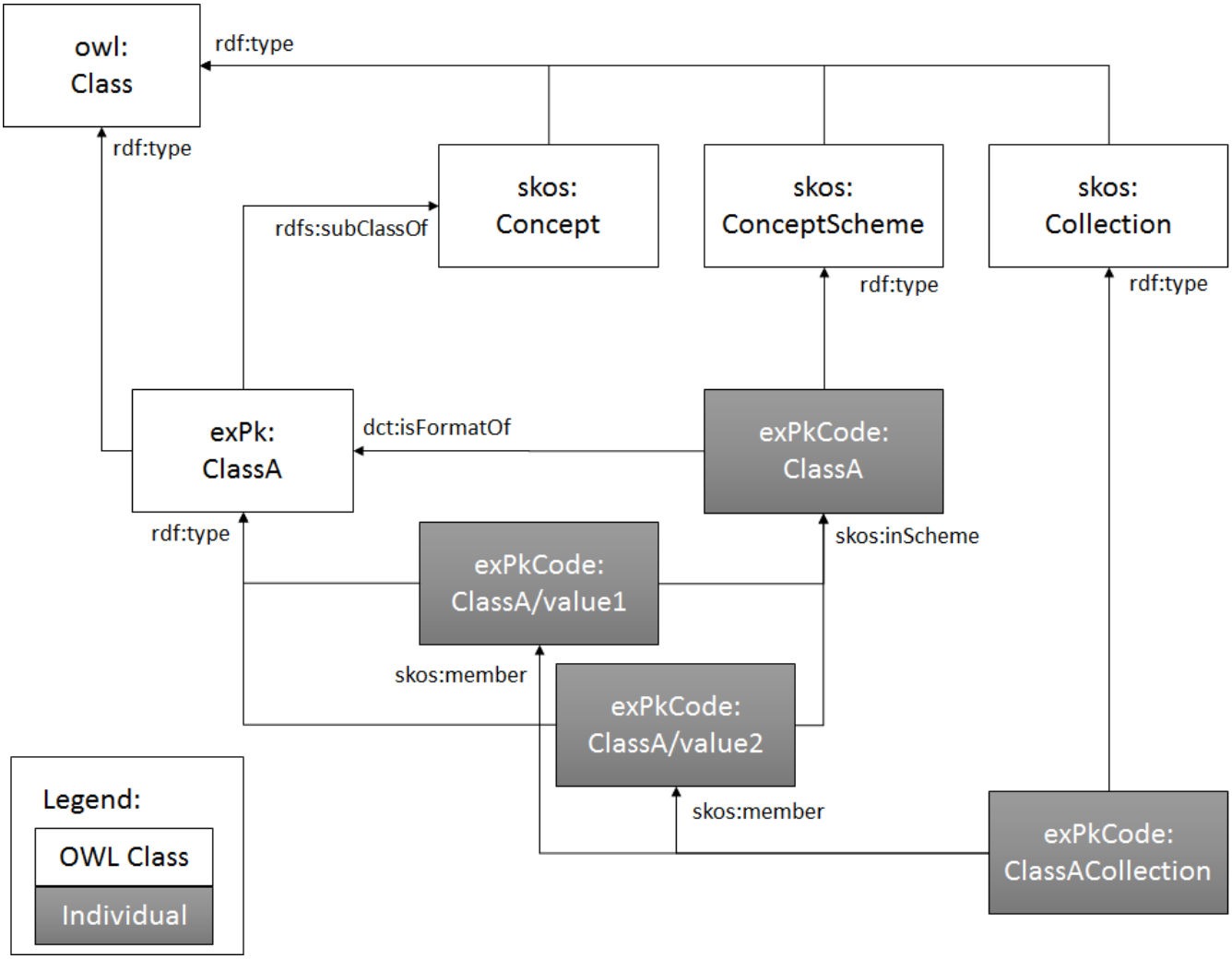

Figure 9. Conversion of <<codelist>> UML classes as defined by ISO 19150-2

This rule can be augmented as follows:

- If *rule-owl-cls-codelist-19150-2-skos-collection* is enabled, then a SKOS collection is created.
  - NOTE: Omitting the SKOS collection may be desirable for a community, because all concepts belonging to a particular scheme can be identified using their skos:inScheme relations. SKOS collections are primarily useful for grouping concepts - especially a subset of the concepts in a concept scheme - under a common label.
- By setting the *codeNamespace* configuration parameter to a specific URI, the rules from ISO 19150-2 (section 6.2.7, table 8) to construct the *codeNamespace* can be overridden. This is useful if the code namespace chosen by a community does not match the structure defined by ISO 19150-2.
  - Note that this would result in a single namespace for all code lists contained in the application schema that are converted by *rule-owl-cls-codelist-19150-2*, which is suitable if *rule-owl-pkg-singleOntologyPerSchema* is enabled.
  - The *codeNamespaceForEnumerations* configuration parameter can be used to set a specific namespace URI for enumerations when encoding them like codelists under *rule-owl-cls-enumerationAsCodelist*.
- If *rule-owl-cls-codelist-19150-2-objectOneOfForEnumeration* is enabled, and the code list actually is an <<enumeration>> that is treated like a code list under *rule-owl-cls-enumerationAsCodelist*,

then an OWL ObjectOneOf class expression is defined for the OWL class representing the code list. The expression enumerates the named individuals representing the code values and thus expresses the semantics of the enumeration as a closed list, in a standard way.

- If *rule-owl-cls-codelist-19150-2-owlClassInDifferentNamespace* is enabled, then the OWL class that is derived from the code list according to ISO 19150-2 is assigned to an RDF namespace defined by the configuration parameter *codeListOwlClassNamespace*. Otherwise the OWL class will be assigned to the application schema namespace, as in ISO 19150-2.
  - If the namespace of the application schema ontology can change between subsequent versions of the ontology, we recommend moving the OWL class representing the code list to a namespace that is different from the application schema ontology namespace. This is explained in more detail further below.
  - The *codeListOwlClassNamespaceForEnumerations* configuration parameter can be used to set a specific namespace URI for the OWL class derived from enumerations when encoding them like codelists under *rule-owl-cls-enumerationAsCodelist*.
- The name of the SKOS concept scheme can be augmented by appending a suffix, which can be defined via the optional configuration parameter *skosConceptSchemeSuffix*.
  - If the *codeListOwlClassNamespace* and the *codeNamespace* / *codeNamespaceForEnumerations* parameters are set to the same value, the OWL class and the SKOS individuals representing the code list or enumeration (as code list under *rule-owl-cls-enumerationAsCodelist*) would be in the same namespace, resulting in a name conflict between the OWL class and the SKOS concept scheme. This conflict can be solved with parameter *skosConceptSchemeSuffix*.
- If *rule-owl-cls-codelist-19150-2-differentIndividuals* is enabled, then an individual inequality axiom is defined for the named individuals that represent the codes of the code list. This axiom ensures that a reasoner will never consider these individuals as equal. Without this axiom, a reasoner can treat individuals as equal because the semantics of OWL 2 does not make the unique name assumption — that is, it does not assume distinct individuals to be necessarily different.
- If *rule-owl-cls-codelist-19150-2-conceptSchemeSubclass* is enabled, then another OWL class representing the code list is created. It is a subclass of skos:ConceptScheme. If the tagged value *skosConceptSchemeSubclassName* is provided for the code list, then it is used as the name for this class. Otherwise, its name is the name of the code list (potentially augmented via parameter *skosConceptSchemeSubclassSuffix*). The named individual that would otherwise be created as a type of skos:ConceptScheme will then be created using the subclass of skos:ConceptScheme as type. This additional subclass can be useful in case there is a need to differentiate between different kinds of concept schemes.

Considerations on Ontology Lifecycle and Code List Conversion

An application schema can evolve, meaning that multiple versions of the schema can be published. An automatically derived ontology can then also have multiple versions. ISO 19150-2 defines rules for the conversion of code lists to OWL and SKOS. This section discusses potential issues that can arise when considering multiple versions of an application schema and the derived ontology, specifically in these cases:

- The ontology name changes between subsequent versions of the ontology (see NOTE below).
- The code lists defined by the application schema are externalized, meaning they are converted to OWL and SKOS only once; afterwards, the OWL and SKOS representation is managed outside of the application schema, which subsequently only refers to this representation.

NOTE

If ISO 19150-2 is strictly followed, the ontology name can change if the *URIbase* changes or if the names of UML packages are changed. The conversion rules described in the section on package name and namespaces can also lead to changes in the names of ontologies derived from different versions of the application schema namespaces.

Assume that an application schema defines a code list *ClassA* that contains a collection of codes. The intent is to derive an ontology from this schema, thereby deriving an OWL and SKOS representation of the code list (via *rule-owl-cls-codelist-19150-2*). Also assume that this representation will subsequently be managed outside of the application schema. For example, addition of new codes as well as deprecation of old codes may be performed in a registry, without reflecting those changes in the application schema. To acknowledge the fact that the code list is managed externally, subsequent versions of the application schema only contain a stub for the code list.

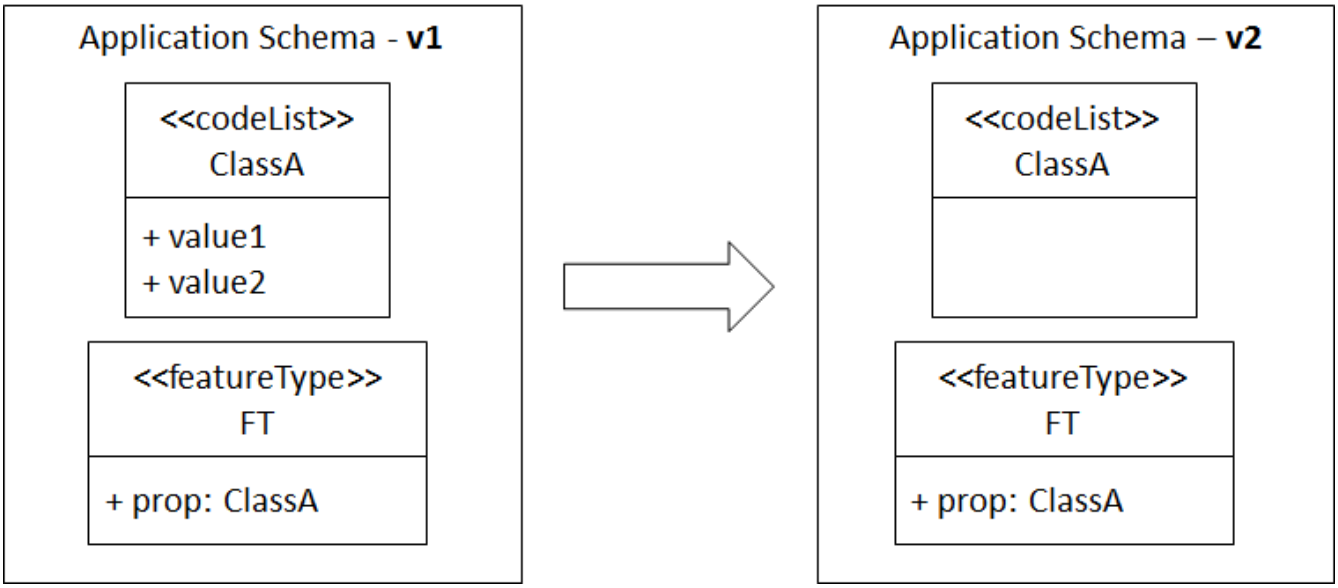

Figure 10. Externalizing code lists - Overview of the evolution of the example application schema

Consider what happens if *rule-owl-cls-codelist-19150-2* (following ISO 19150-2, table 16) is used to convert <<codeList>> ClassA in both versions of the application schema. This would lead to a disconnect as shown in the following figure.

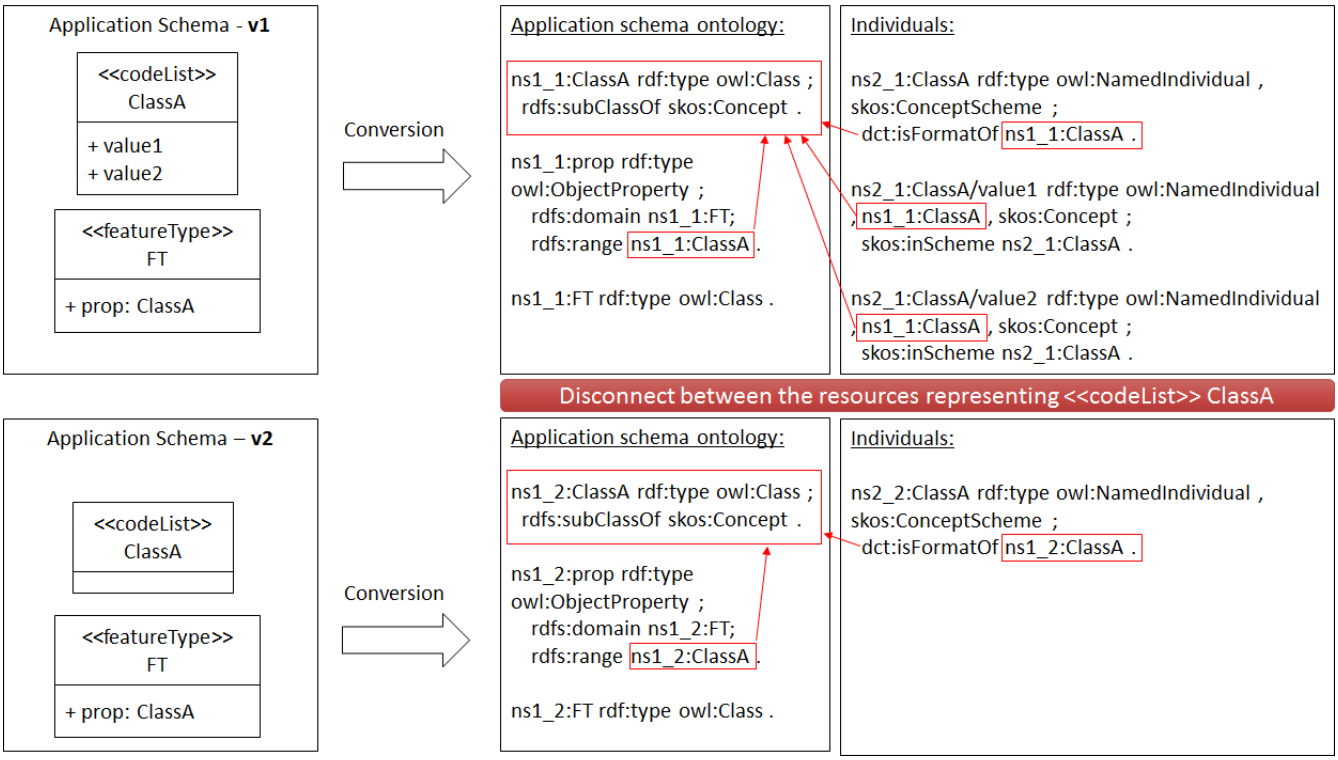

Figure 11. Externalizing code lists - Disconnect

The code list is converted into two independent sets of OWL classes and SKOS individuals. The namespaces of OWL class and SKOS concept scheme changed in the second version of the ontology, which is indicated by different namespace prefixes.
The second concept scheme is meaningless because it does not have any concepts. One could add additional statements to assert equivalence of the OWL classes ns1_1:ClassA and ns1_2:ClassA as well as SKOS concept schemes ns2_1:ClassA and ns2_2:ClassA. However, that would have to be done each time the namespace changes in a new version of the ontology. Also, the link between a property and the actual SKOS concepts that can be used as values would be obscured.

To avoid this disconnect, the "stub" code list in the second version of the application schema should be converted using *rule-owl-cls-codelist-external*.

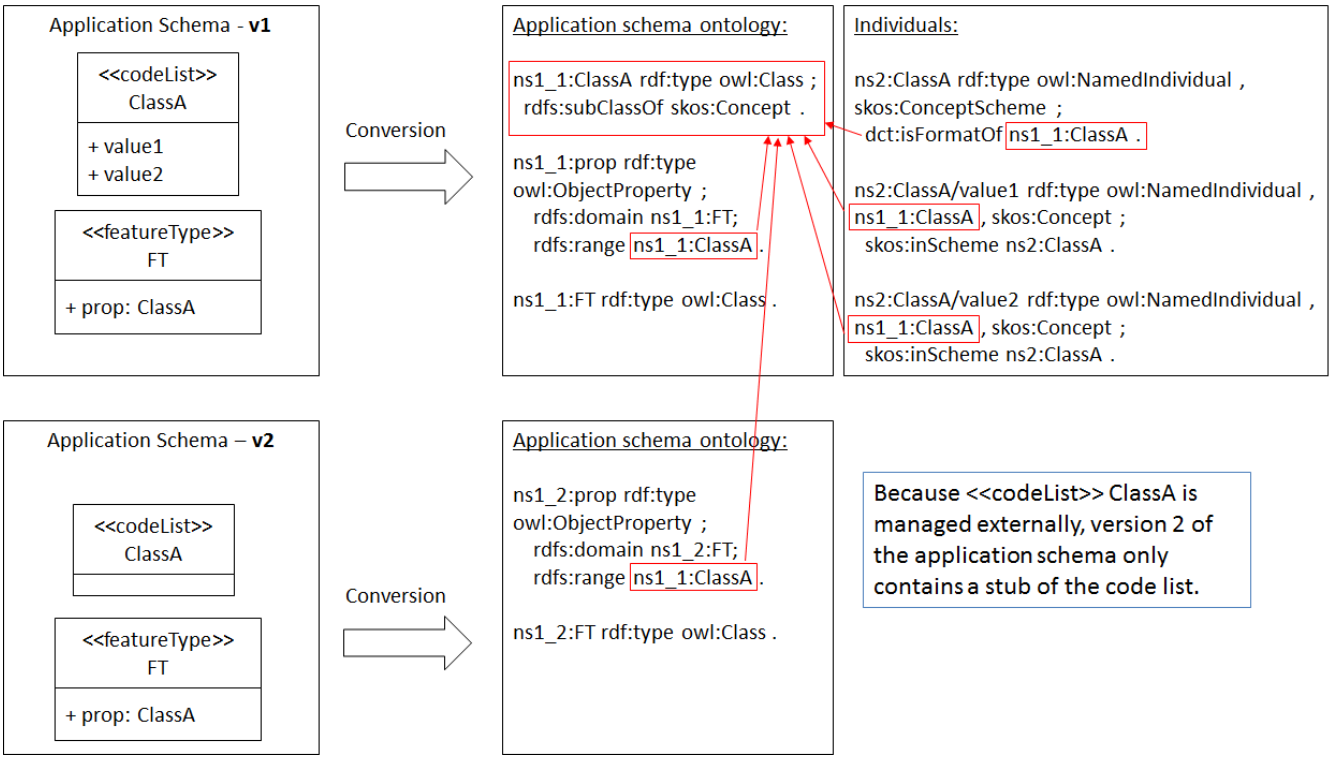

Figure 12. Externalizing code lists - OWL class in application schema ontology namespace

NOTE The SKOS collection has been omitted in the figure for simplicity.

In this case (illustrated in the figure above), the next version of the ontology (derived from application schema v2) no longer defines 'ClassA'. Instead, property 'prop' references the OWL 'ClassA' that has initially been derived from the code list (this reference can be achieved by adding an *RdfTypeMapEntry* for the code list to the ShapeChange configuration). Values of 'prop' can thus be individuals of 'ClassA'. The disconnect described before no longer exists.
However, this requires an import of the previous ontology. To avoid this, the OWL 'ClassA' should have been created in a namespace that is different to that of the application schema (using the configuration parameter *codeListOwlClassNamespace*). This approach is illustrated in the following figure.

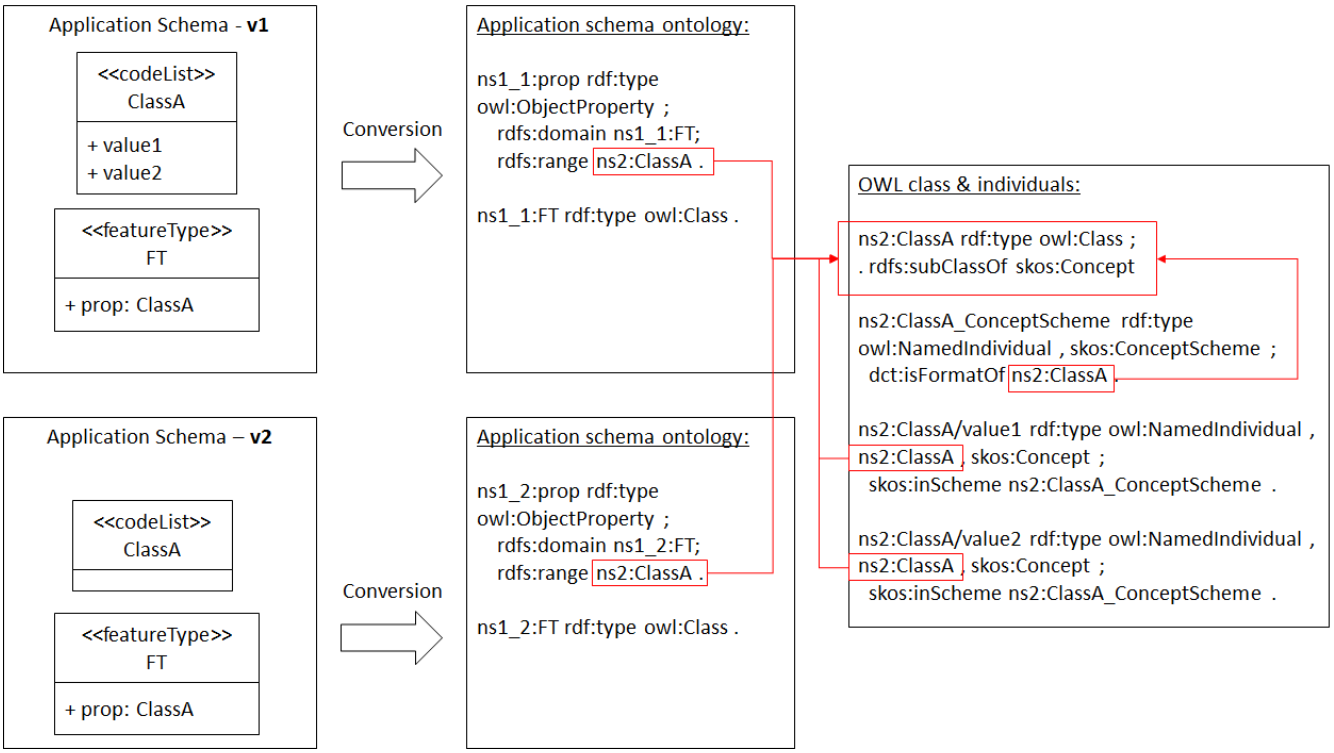

Figure 13. Externalizing code lists - OWL class outside of application schema ontology namespace (recommended approach)

If code lists are *not* managed outside the application schema, the situation is different. Enumerations encoded like code lists are a good example:

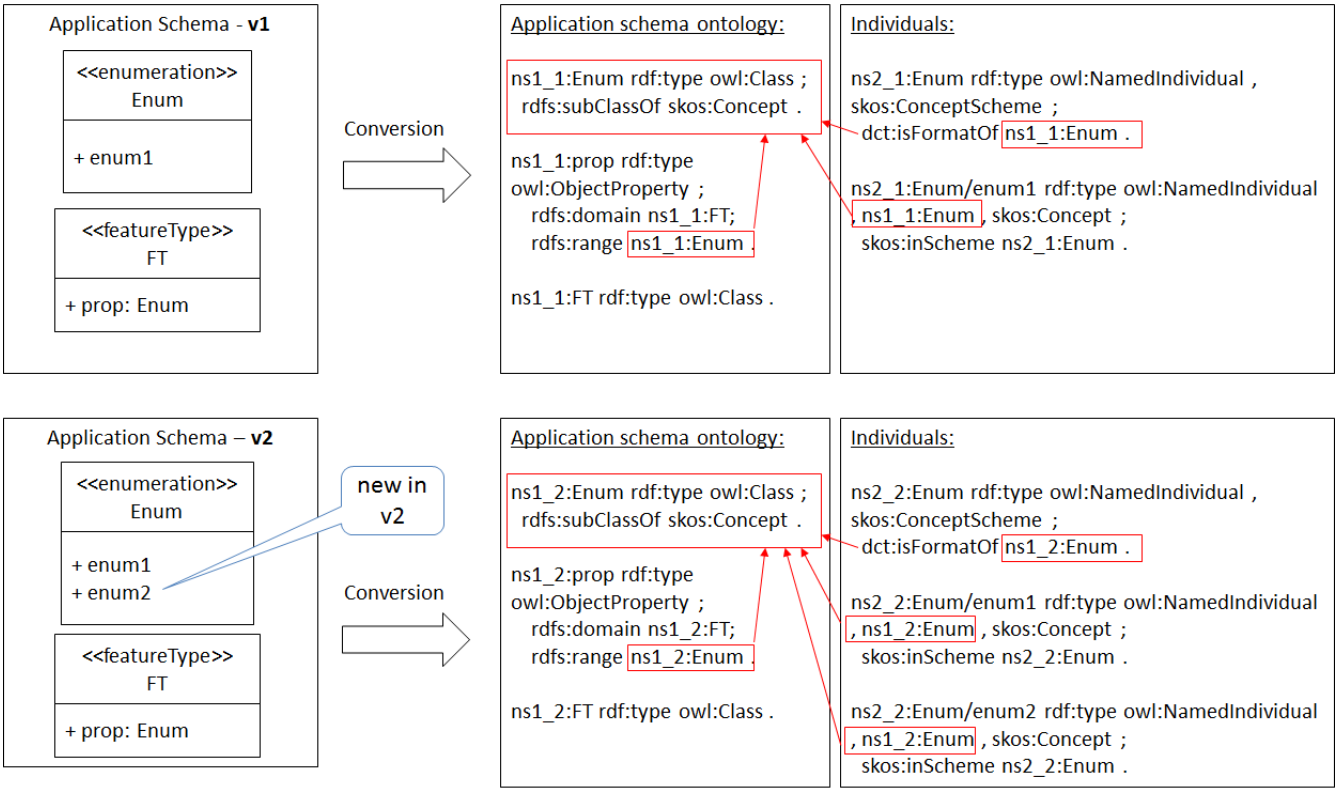

Figure 14. Enumerations as code lists - managed in the application schema

Both versions of the UML application schema contain the complete enumeration. The only difference is a new listed value added in v2.

The enumeration is converted using *rule-owl-cls-codelist-19150-2* - both in the first and the second

version of the ontology. Consequently, both ontologies contain their own OWL class representing the code list. In each version, the OWL class representing the code list is referenced by a set of SKOS individuals (concepts and concept scheme) derived from the same version. There is no disconnect because each application schema version defines the relevant codes, which are represented by the OWL and SKOS resources that are produced when creating the respective ontologies.

Code Hierarchies

Some communities define hierarchically structured code lists. These code lists contain broader and narrower terms.

For example, the list of codes to denote a feature function of a building can be structured as follows:

- accommodation
  - longTermAccommodation
    - dependentsHousing
    - dormitory
    - residence
  - shortTermAccommodation
    - communalBath
    - guestHouse
    - hostel
- agriculture
  - grazing
  - growingOfCrops
  - mixedFarming
  - raisingOfAnimals

Here, the term "accommodation" is broader than "longTermAccommodation" and "shortTermAccommodation". Likewise, the term "agriculture" is broader than "grazing" and "growingOfCrops". However, there is no broader term for "accommodation" or "agriculture.

If the broader/narrower relationship between terms is known, they can be represented in SKOS using the skos:broader and/or skos:narrower relations. It is also possible to identify the terms that are at the top of the hierarchy, using the skos:topConceptOf property.

ISO 19150-2 does not cover the conversion of hierarchically structured code lists, presumably because ISO 19103 - where the <<codeList>> stereotype is defined - does not specify how this structure should be modeled (there is only an example in ISO 19103 Annex G.5.3).

If *rule-owl-prop-code-broader-byBroaderListedValue* is enabled, then skos:broader relationships are established for code values as follows: If a code - e.g. "longTermAccommodation" - has the tag "broaderListedValue" and it contains the name of another code - e.g. "accommodation" - from the

same code list, then the following triple is added to the ontology: "longTermAccommodation" skos:broader "accommodation". If a code does not have a value for the tag "broaderListedValue" then a skos:topConceptOf relationship is added to the SKOS concept that represents the code, referencing the SKOS concept scheme that the concept is skos:inScheme of. If the value of the tag "broaderListedValue" does not match any code in the code list, then ShapeChange logs a warning and treats it as a top concept.

The following listing gives an example of converting the NSG *ApronAccessibilityStatusType*, where the enums *lockedOpen* and *lockedClosed* have enum *locked* as "broaderListedValue".

```
<owl:Class rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType">
 <owl:oneOf rdf:parseType="Collection">
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/closed">
   <skos:topConceptOf>
    <skos:ConceptScheme
     rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme">
     <skos:prefLabel xml:lang="en">Apron Accessibility Status Type - Concept
Scheme</skos:prefLabel>
     <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=106468"/>
     <skos:definition xml:lang="en">![CDATA[Definition: A coded domain value denoting
the
      accessibility status type of an apron. Description: [None
Specified]]]</skos:definition>
     <rdfs:label xml:lang="en">ApronAccessibilityStatusType_ConceptScheme</rdfs:label>
     <dct:isFormatOf
      rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType"/>
    </skos:ConceptScheme>
   </skos:topConceptOf>
   <skos:inScheme
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Closed</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116678"/>
   <skos:definition xml:lang="en">![CDATA[Definition: Access is officially prohibited.
Description:
    May be covered and/or blocked by a physical barrier.]]</skos:definition>
   <rdfs:label xml:lang="en">closed</rdfs:label>
  </e:ApronAccessibilityStatusType>
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/limited">
   <skos:topConceptOf
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:inScheme
```

```
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Limited</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116677"/>
   <skos:definition xml:lang="en">![CDATA[Definition: A limitation on access, but not
function, has
    been imposed. Description: Not necessarily enforced by a physical
barrier.]]</skos:definition>
   <rdfs:label xml:lang="en">limited</rdfs:label>
  </e:ApronAccessibilityStatusType>
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/locked">
   <skos:topConceptOf
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:inScheme
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Locked</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116672"/>
   <skos:definition xml:lang="en">![CDATA[Definition: Access is prevented by a
physical barrier,
    requiring special means to pass (for example: a key). Description: [None
    Specified]]]</skos:definition>
   <rdfs:label xml:lang="en">locked</rdfs:label>
  </e:ApronAccessibilityStatusType>
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/lockedClosed">
   <skos:broader
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/locked"/>
   <skos:inScheme
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Locked Closed</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116674"/>
   <skos:definition xml:lang="en">![CDATA[Definition: Access is officially prohibited
and is
    restricted by a physical barrier, requiring special means to pass (for example: a
key).
    Description: [None Specified]]]</skos:definition>
   <rdfs:label xml:lang="en">lockedClosed</rdfs:label>
  </e:ApronAccessibilityStatusType>
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/lockedOpen">
   <skos:broader
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/locked"/>
```

```
   <skos:inScheme
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Locked Open</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116675"/>
   <skos:definition xml:lang="en">![CDATA[Definition: Access is officially allowed
although
    restricted by a physical barrier that is currently open, requiring special means
to close and
    prevent future passage (for example: a key). Description: [None
Specified]]]</skos:definition>
   <rdfs:label xml:lang="en">lockedOpen</rdfs:label>
  </e:ApronAccessibilityStatusType>
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/open">
   <skos:topConceptOf
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:inScheme
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Open</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116673"/>
   <skos:definition xml:lang="en">![CDATA[Definition: Access is officially allowed.
Description: May
    be covered and/or blocked by a physical barrier that is temporarily
    passable.]]</skos:definition>
   <rdfs:label xml:lang="en">open</rdfs:label>
  </e:ApronAccessibilityStatusType>
  <e:ApronAccessibilityStatusType
   rdf:about="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType/restricted">
   <skos:topConceptOf
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:inScheme
    rdf:resource="http://api.nsgreg.nga.mil/ontology/neo-
enum/8.0#ApronAccessibilityStatusType_ConceptScheme"/>
   <skos:prefLabel xml:lang="en">Restricted</skos:prefLabel>
   <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=116676"/>
   <skos:definition xml:lang="en">![CDATA[Definition: Access is officially allowed
although a
    limitation on function has been imposed. Description: Not necessarily enforced by
a physical
    barrier.]]</skos:definition>
   <rdfs:label xml:lang="en">restricted</rdfs:label>
  </e:ApronAccessibilityStatusType>
 </owl:oneOf>
 <rdfs:subClassOf rdf:resource="http://www.w3.org/2004/02/skos/core#Concept"/>
 <skos:prefLabel xml:lang="en">Apron Accessibility Status Type</skos:prefLabel>
```

```
 <rdfs:isDefinedBy rdf:resource="http://nsgreg.nga.mil/as/view?i=106468"/>
 <skos:definition xml:lang="en">![CDATA[Definition: A coded domain value denoting the
accessibility
  status type of an apron. Description: [None Specified]]]</skos:definition>
 <rdfs:label xml:lang="en">ApronAccessibilityStatusType</rdfs:label>
</owl:Class>
```

Rule Execution Priority and Dependencies

The conversion rules for code lists are executed with the following priority:

1. *rule-owl-cls-codelist-external*
2. *rule-owl-cls-codelist-19150-2*
    - The following conversion rule(s) can be used with this rule:
        - *rule-owl-cls-codelist-19150-2-skos-collection*
        - *rule-owl-cls-codelist-19150-2-objectOneOfForEnumeration*
        - *rule-owl-cls-codelist-19150-2-differentIndividuals*
        - *rule-owl-cls-codelist-19150-2-owlClassInDifferentNamespace*
        - *rule-owl-cls-codelist-19150-2-conceptSchemeSubclass*
        - *rule-owl-prop-code-broader-byBroaderListedValue*

Additional rules may be added in the future.

NOTE

Because the conversion rules are not mutually exclusive, they can be combined in an encoding rule. An application schema can therefore have code lists that are used as stubs (to reference an external resource representing the code list), as well as code lists that contain actual codes. This can be useful when:

- The application schema contains some code lists that are managed by other organizations. These code lists would only be referenced in the application schema, using *rule-owl-cls-codelist-external* to encode them.
- The application schema also contains code lists with actual codes, for which an encoding (to OWL and SKOS) is not available yet and thus shall automatically be derived by ShapeChange, for example using *rule-owl-cls-codelist-19150-2*.
    - Once the encoding is available, the <<codeList>> UML classes can be transformed into stubs to reference the externally managed encoding. This approach supports the transition to an application schema where all code lists used by the schema are managed externally. In this case, it is recommended that *rule-owl-cls-codelist-19150-2-owlClassInDifferentNamespace* be used during the initial conversion of the code list.

Other Types

At the moment, no specific conversion rules are defined for other types.

8.2.4. Property

General

The conversion of UML properties (attributes and navigable association roles) into OWL properties is enabled or disabled by respectively including or excluding *rule-owl-prop-general* in the encoding rule.

However, this rule does not apply to UML properties of <<codeList>> and <<enumeration>> types, because the rules defined to convert those types specify if and how their properties - i.e., listed values - are converted.

NOTE	Being able to "switch off" property conversion is useful when deriving a taxonomy instead of a complete ontology from the application schema.

Under *rule-owl-prop-general*, a UML property is converted to:

- an OWL object property, if the value type of the UML property is represented as an OWL class;
- an OWL datatype property, if the value type of the UML property is represented as an OWL datatype or literal.

However, in general, if an RdfPropertyMapEntry in the ShapeChange configuration provides a mapping to an RDF/OWL property for a UML property from the processed schema, then the UML property is not encoded. Instead, whenever the UML property is used in the schema, the RDF/OWL property specified by the map entry is used in the RDF implementation.

For example, the following configuration fragment specifies that:

- UML property "attCommon" (from any UML type in the schemas selected for processing by ShapeChange) will be mapped to the target RDF/OWL property http://example.org/1#prop11.
  - This supports general mapping of named properties.
- When converting "attCommon", and if the range of the target property http://example.org/1#prop11 is needed (for example in existential and universal quantifications, as well as quantified property cardinality restrictions), then http://example.org/1#ClassA is used as range.
  - An RdfPropertyMapEntry that maps a UML property to a target RDF/OWL property can be used to define a "range" for the target property.
    This is useful if the declaration of the external property does not include a range axiom, or if the range shall be set on a case-by-case basis. For example, the range of the target RDF/OWL property "location" could be any geometry type. When "location" is used to represent the property of a specific UML class from the application schema, for example "TransportationStop", a single geometry type may be more appropriate, for example "Point".
- UML property "att12" (from UML type "Feature102") will be mapped to RDF property

http://example.org/1#prop11.

- This supports precise mappings of individual UML properties (scoped to a specific class).
- The range to be used for http://example.org/1#prop11 when it represents Feature102::att12 is http://example.org/1#ClassB.

```
  <sc:rdfMapEntries>
    <sc:RdfPropertyMapEntry property="attCommon" target="ex1:prop11"
range="ex1:ClassA"/>
    <sc:RdfPropertyMapEntry property="Feature102::att12" target="ex1:prop11"
range="ex1:ClassB"/>
  </sc:rdfMapEntries>
  <sc:namespaces>
    <sc:Namespace nsabr="ex1" ns="http://example.org/1#"/>
  </sc:namespaces>
```

The XML Schema definition of rdfMapEntries (including documentation of attributes) is provided in Annex B.

NOTE

- An RdfPropertyMapEntry is used to describe the mapping of application schema properties to RDF/OWL properties.
- An RdfPropertyMapEntry can be used to define the range of the RDF/OWL property, which may be especially useful when using it to implement a specific UML property.
- If a UML property P is mapped to an RDF/OWL property from an external ontology EX, then the resulting ontology RES will import EX. The property declaration from EX will then be available and used in RES. RES will not contain an OWL property declaration of P in its own namespace.
- No 'targetType' attribute is defined in the XML Schema of RdfPropertyMapEntry. As in an RdfTypeMapEntry, such an attribute could indicate whether the 'target' is an RDF property, an OWL object property, or an OWL datatype property. Currently, this information is not required for the conversion to RDF/OWL/SKOS. This extension can be added in the future.

Property Name

Following ISO 19150-2, the OWL property name is a combination of the RDF namespace of the ontology, the UML class name (conditional), and the UML property name:

```
propertyName = rdfNamespace [umlClassName "."] propertyLocalName
propertyLocalName = umlAttributeName / umlRoleName
```

The property local name is given in lower camel case. Punctuation characters other than dash and underscore are replaced by underscore characters. Space characters are removed.

The UML class name is included if the property is not converted to a globally scoped property (see

next topic).

This naming convention is implemented in *rule-owl-prop-iso191502-naming*.

NOTE	At the moment, *rule-owl-prop-iso191502-naming* is the default for creating the name of an OWL property. Additional rules can be added in the future to implement alternative naming conventions.

Scope - Local vs. Global

In UML, an attribute belongs to the class that defines it. Likewise, an association role is a property that belongs to a specific class. In RDF, a property can be described in terms of the class to which it applies (as its domain), but it can also be described independently of any class.

In this respect, there is a mismatch between UML and RDF; in UML, a property belongs to its class, while in RDF, a property can be used by multiple classes. In other words, in UML, a property is always scoped to a specific class, while in RDF the scope of properties can be global.

NOTE	A property domain axiom is defined for an RDF property with local scope. No such axiom is defined for an RDF property with global scope, allowing it to be used by multiple classes.

A straightforward solution for converting UML properties would be to convert each UML property to a uniquely corresponding OWL property, relating it to a specific class by:

1. adding the class name that the property belongs to in UML to the OWL property name (so that the OWL property declaration is unique in the ontology), and
2. declaring the OWL representation of its owning UML class as the domain of the OWL property.

This approach is covered by the conversion rule *rule-owl-prop-localScopeAll*.

However, if multiple properties with the same name and the same semantics exist in an application schema, this localizing approach will lead to repetition that would clearly be undesirable and not in the spirit of RDF. In this case, there should be a way to identify which properties can be mapped to a global property definition. Global properties can be reused within a single ontology and across multiple ontologies.

The following rules are available to identify and encode UML properties as global OWL properties:

Common, Unique Attribute Concepts Defined in a Dictionary

Some communities specify most (if not all) schema concepts within a dictionary. In addition to concepts for classes, the dictionary also contains concepts for each attribute that occurs within the schema. A specific class concept in the application schema uses a particular set of common attribute concepts defined in the dictionary. Given such a framework, all UML attributes can be encoded as globally scoped OWL properties. This behavior will be executed if *rule-owl-prop-globalScopeAttributes* is enabled.

Explicit Identification

If *rule-owl-prop-globalScopeByConversionParameter* is enabled, then conversion parameters contained in the OWL target of the ShapeChange configuration file specify which properties from the conceptual schema in UML shall be represented as global OWL properties.

For example, the following configuration fragment specifies that:

- "att4" from "Feature1" - which belongs to the application schema with package name "Schema 1" - will be converted to a globally scoped OWL property.
  - All other "att4" UML properties from "Schema 1" will be implemented using the OWL implementation of this global property.
- UML property "att" (from UML type "Feature2" in "Schema 2") is converted to a globally scoped OWL property.
  - Other "att" UML properties will be converted to locally scoped OWL properties (because an additional conversion parameter as for properties with name "att4" is not defined).
    - NOTE: if the schema contains more properties with name "att" and similar semantics, then they should be mapped to the globally scoped property.
  - The UML properties "att11" and "att12" (from UML types "Feature3" and "Feature4" - both from "Schema 3") will be implemented using the OWL implementation of this global property.

```
  <sc:rdfConversionParameters>
    <sc:PropertyConversionParameter property="Feature1::att4" schema="Schema 1"
global="true"/>
    <sc:PropertyConversionParameter property="att4" schema="Schema 1"
target="Feature1::att4" schema="Schema 1"/>
    <sc:PropertyConversionParameter property="Feature2::att" schema="Schema 2"
global="true"/>
    <sc:PropertyConversionParameter property="Feature3::att11" schema="Schema 3"
target="Feature2::att" schema="Schema 2"/>
    <sc:PropertyConversionParameter property="Feature4::att12"  schema="Schema 3"
target="Feature2::att" schema="Schema 2"/>
  </sc:rdfConversionParameters>
```

The XML Schema definition of rdfConversionParameters (including documentation of attributes) is provided in Annex B.

NOTE

- If a UML property is the subject of both a PropertyConversionParameter and an RdfPropertyMapEntry, then the former is ignored and ShapeChange logs a warning.
- Setting "global" to "true" is only allowed if the "property" identifies the property of a specific UML class from a specific schema. This way, there cannot be ambiguity regarding the semantics of the globally scoped property - it is defined by a single UML property from the model.
- "global" has no effect if "target" is also present in the same conversion parameter - because then the UML property will be mapped to another UML property that is implemented as a global property.
- If the PropertyConversionParameter has a "target" (which shall be scoped to a class to avoid ambiguity when a schema is encoded as multiple ontologies) then it must also have a "targetSchema", to unambiguously identify a global property to which the "property" from the PropertyConversionParameter is mapped.
- Currently, there is no mechanism to encode a specific UML property as a global RDF/OWL property with a name that is different than the one computed from the name of the UML property.
- The combination of "global" and "subPropertyOf" is allowed. In that case, a "subPropertyOf" declaration is added to the encoding of the global OWL property.

Unique Property Name in Ontology

If *rule-owl-prop-globalScopeByUniquePropertyName* is enabled, then ShapeChange will determine if the name of a UML property from the application schema is going to be unique within the ontology into which its OWL property representation will be placed. If it is unique, then the property will be converted to a globally scoped property.

NOTE This approach is suggested by ISO 19150-2 (section 6.2.6).

Comparing Global and Mapped Property

If a property is mapped to a global one, ShapeChange can compare the range, definition, description, and alias of the two properties. This can be useful to identify potential inconsistencies that may be introduced through the mapping. A warning will be logged for each potential inconsistency. This comparison is implemented in *rule-owl-prop-mapping-compare-specifications*.

Rule Execution Priority and Dependencies

The conversion rules to determine the scope of OWL properties are executed with the following priority:

1. *rule-owl-prop-localScopeAll*
2. *rule-owl-prop-globalScopeAttributes*
3. *rule-owl-prop-globalScopeByUniquePropertyName*

4. *rule-owl-prop-globalScopeByConversionParameter*
    - If both *rule-owl-prop-globalScopeAttributes* and *rule-owl-prop-globalScopeByConversionParameter* (or *rule-owl-prop-globalScopeByUniquePropertyName*) are enabled in an encoding rule, then for the conversion of UML attributes there is no need to check the conversion parameters (or property names), because all UML attributes will be converted to globally scoped properties.

Additional rules may be added in the future.

NOTE	ISO 19150-2 (section 6.2.6) also suggests that properties with same "meanings" can be converted to globally scoped properties. However, this is a rather vague definition. Therefore, the design does not include a corresponding conversion rule.

Range

Identifying the Range of a Property

A UML property from the application schema is either encoded as an OWL property in the application schema ontology, or mapped to an RDF/OWL property (using an RdfPropertyMapEntry).

If the UML property is encoded as an OWL property in the application schema ontology, then the range of the OWL property is the RDFS (or OWL) class or datatype (identified by IRI) that represents the value type of the UML property.

If the UML property is mapped to an RDF/OWL property, then the range of the RDF/OWL property applies. The range is either given explicitly by the RdfPropertyMapEntry, or it is undefined. In that case, statements that require the range of the RDF/OWL property cannot be made. If such a situation occurs, ShapeChange will log a warning. This is not necessarily an error, because the external ontology may define a range axiom for the property, thus the range would be defined globally.

Range Definition - Global vs. Local

The range of a property can be encoded globally or locally:

- If *rule-owl-prop-range-global* is enabled, then an OWL range axiom (Object-/DataPropertyRange) is defined for the OWL property that represents the UML property.
- If *rule-owl-prop-range-local-withUniversalQuantification* is enabled, then OWL universal quantifications are included in the ontology, to restrict the range of a property only within the scope of those OWL classes which represent UML classes that own the property.

Declaring the range locally can be desirable with respect to re-use of properties in other ontologies. In such a case, the other ontology may prefer to use a specific range for the property. If the range is already globally defined for an OWL property, then it cannot easily be changed.

NOTE	Global or local range must not be confused with global or local scope of a property. The former controls whether an OWL range axiom is declared for an OWL property, while the latter controls the naming of the OWL property and whether a domain axiom is declared for it.

Multiplicity

As described in ISO 19150-2, section 6.9, the multiplicity of a UML property specifies how many values the property can have. The default multiplicity is 1.

In OWL, the cardinality of a property need not be restricted. This means that an instance of a class may have zero or any number of values for a given property. However, cardinality restrictions for properties can be expressed in OWL.

Object and *Data Property Cardinality Restrictions* are available to restrict the number of times that a property can be assigned to a class. They can be qualified and unqualified.

A qualified cardinality restriction only applies to property values that are of a specific type.

```
ex:ownsAnimal rdf:type owl:ObjectProperty ;
    rdfs:range ex:Animal .

ex:Animal rdf:type owl:Class .

ex:Elephant rdf:type owl:Class ;
    rdfs:subClassOf ex:Animal .

ex:Zoo rdf:type owl:Class ;
  rdfs:subClassOf [ rdf:type owl:Restriction ;
    owl:onProperty ex:ownsAnimal ;
    owl:minCardinality "1"^^xsd:nonNegativeInteger ;
  ] .

ex:SmallZoo rdf:type owl:Class ;
    rdfs:subClassOf ex:Zoo ,
        [ rdf:type owl:Restriction ;
            owl:onProperty ex:ownsAnimal ;
            owl:maxQualifiedCardinality "10"^^xsd:nonNegativeInteger ;
            owl:onClass ex:Elephant
        ] .

ex:BigZoo rdf:type owl:Class ;
    rdfs:subClassOf ex:Zoo ,
        [ rdf:type owl:Restriction ;
            owl:onProperty ex:ownsAnimal ;
            owl:minQualifiedCardinality "11"^^xsd:nonNegativeInteger ;
            owl:onClass ex:Elephant
        ] .
```

An unqualified cardinality restriction is sufficient to state that a Zoo must own at least one animal. The example shows how qualified cardinality restrictions can be used to specify that a Zoo can be classified as a SmallZoo if it owns at most ten elephants, and as a BigZoo if it owns at least eleven elephants.

NOTE A reasoner could infer from the cardinality restrictions stated in this example that SmallZoo and BigZoo are disjoint classes.

Two conversion rules are available to convert the multiplicity of a UML property into OWL property cardinality restrictions (which are defined on the OWL class which represents the UML class owning the UML property):

- If *rule-owl-prop-multiplicityAsUnqualifiedCardinalityRestriction* is enabled, then the multiplicity of a UML property is encoded with an unqualified cardinality restriction.
- If *rule-owl-prop-multiplicityAsQualifiedCardinalityRestriction* is enabled, then the multiplicity of a UML property is encoded with a qualified cardinality restriction.

NOTE ISO 19150-2 appears to require that an OWL property restriction, more specifically a universal quantification (encoded in RDF as owl:allValuesFrom), is encoded together with an OWL property cardinality restriction. This should not be required. As explained in the section on property range, the range of an OWL property can be defined globally using a property range axiom but also locally with a universal quantification.

NOTE It is possible to omit the conversion of multiplicity by simply omitting both *rule-owl-prop-multiplicityAsUnqualifiedCardinalityRestriction* and *rule-owl-prop-multiplicityAsQualifiedCardinalityRestriction* from the encoding rule.

Reasons for doing so include, but are not limited to:

- Applications do not support or require the validation of cardinality constraints.
- The ontology derived from the application schema shall serve as a base ontology that defines the relevant concepts (classes and properties) without cardinality restrictions. Cardinality restrictions would be captured in an additional ontology that would import the base ontology. The base ontology would therefore be lightweight, with more axiomatization and formalism provided in the additional ontology. This also allows for re-use of the base ontology in application ontologies that impose different cardinality restrictions for business reasons.

Regarding OWL functional properties:

OWL functional object and data properties could be useful to restrict the cardinality of a property on a global level. These restrictions ensure that the maximum cardinality of a property is always 1, regardless of the OWL class context in which the property may occur.

The OWL 2 structural specification uses *ex:hasFather* as an example of a functional property.

An example of a property that should not be declared as a functional property is *ex:hasChild*.

At this time, however, no conversion rule is defined with which property functionality axioms can be defined for OWL properties.

Custom subPropertyOf Mappings

If required by a specific encoding rule, special rdfs:subPropertyOf declarations can be added to particular properties (globally, or scoped to a specific type and/or schema) in the ontology. This information is kept separate from the UML model; it is provided in the ShapeChange configuration.

For example, the following configuration fragment specifies that:

- UML property "att1" in UML type "Feature1" (in "Schema 1") shall be declared rdfs:subPropertyOf http://example.org/1#PropX.
- All UML properties named "att2", in all UML types of all schemas selected for processing by ShapeChange, shall be declared rdfs:subPropertyOf http://example.org/1#PropX.

```
  <sc:rdfConversionParameters>
    <sc:PropertyConversionParameter property="Feature1::att1" schema="Schema 1"
subPropertyOf="ex1:PropX"/>
    <sc:PropertyConversionParameter property="att2" subPropertyOf="ex1:PropX"/>
  </sc:rdfConversionParameters>
  <sc:namespaces>
    <sc:Namespace nsabr="ex1" ns="http://example.org/1#"/>
  </sc:namespaces>
```

The XML Schema definition of a PropertyConversionParameter element (including documentation of attributes) is provided in Annex B.

NOTE	"subPropertyOf" has no effect if "target" is also present in the PropertyConversionParameter (because then the UML property is implemented by a global property [for which a "subPropertyOf" statement can be made]).

Attribute

The general conversion rules for UML properties apply for the conversion of UML attributes.

No additional specific rules are defined for the conversion of UML attributes.

Association Role

The general conversion rules for UML properties apply for the conversion of UML association roles.

In addition, the following conversion rules are available:

- If the association to which the association role belongs has a name and *rule-owl-prop-iso191502AssociationName* is enabled, then an iso19150-2:associationName annotation is added to the OWL property representing the association role, with the association name as its value.

- If the association to which the association role belongs is bi-directional and *rule-owl-prop-inverseOf* is enabled, then an owl:inverseOf predicate is added to the OWL property representing the association role, with the IRI of the inverse property as its value.
- If the association role is playing the part role in a shared or composite aggregation, and *rule-owl-prop-iso191502Aggregation* is enabled, then an iso19150-2:aggregationType annotation is added to the OWL property representing the association role, with value "partOfSharedAggregation" or "partOfCompositeAggregation" - depending upon the given type of aggregation.

WARNING

If an association role is converted to a globally scoped OWL property, and other association roles with that name are implemented by this OWL property, then the conversion rules described in this section can cause multiple iso19150-2:associationName annotations, iso19150-2:aggregationType annotations, or owl:inverseOf predicates to be defined for the OWL property. In the following figure, the association role named "role" could be converted to a globally scoped property.

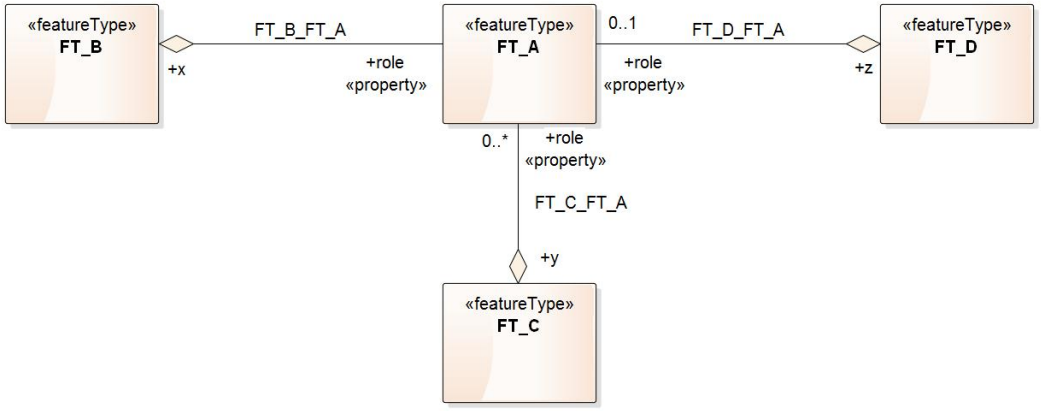

Figure 15. Example of multiple associations with same name for one role

In the case of owl:inverseOf, this can lead to undesired results, because a reasoner would infer that the inverse properties are equivalent.

NOTE

The annotation properties iso19150-2:associationName and iso19150-2:aggregationType are defined as having the domain owl:Class. However, owl:ObjectProperty and owl:DatatypeProperty are not of type owl:Class. Strictly speaking, the annotation properties defined by ISO 19150-2 cannot be used on OWL properties. Also see the discussion on the iso19150-2:constraint annotation property.

8.2.5. Association Class

In UML, an association class has both association and class properties (see UML Superstructure 2.4.1, section 7.3.4). It connects a set of classes and also defines a set of properties that belong to the relationship itself and not to any of the classes.

ISO 19150-2 does not specify conversion rules for association classes.

The OGC GML 3.3 standard defines a rule for converting an association class from a conceptual

schema in UML to GML (see GML 3.3, section 12.3). The rule specifies the mapping of the characteristics of the UML association class into parts of a new structure having an intermediate class associated separately to the two classes originally linked by the association class.

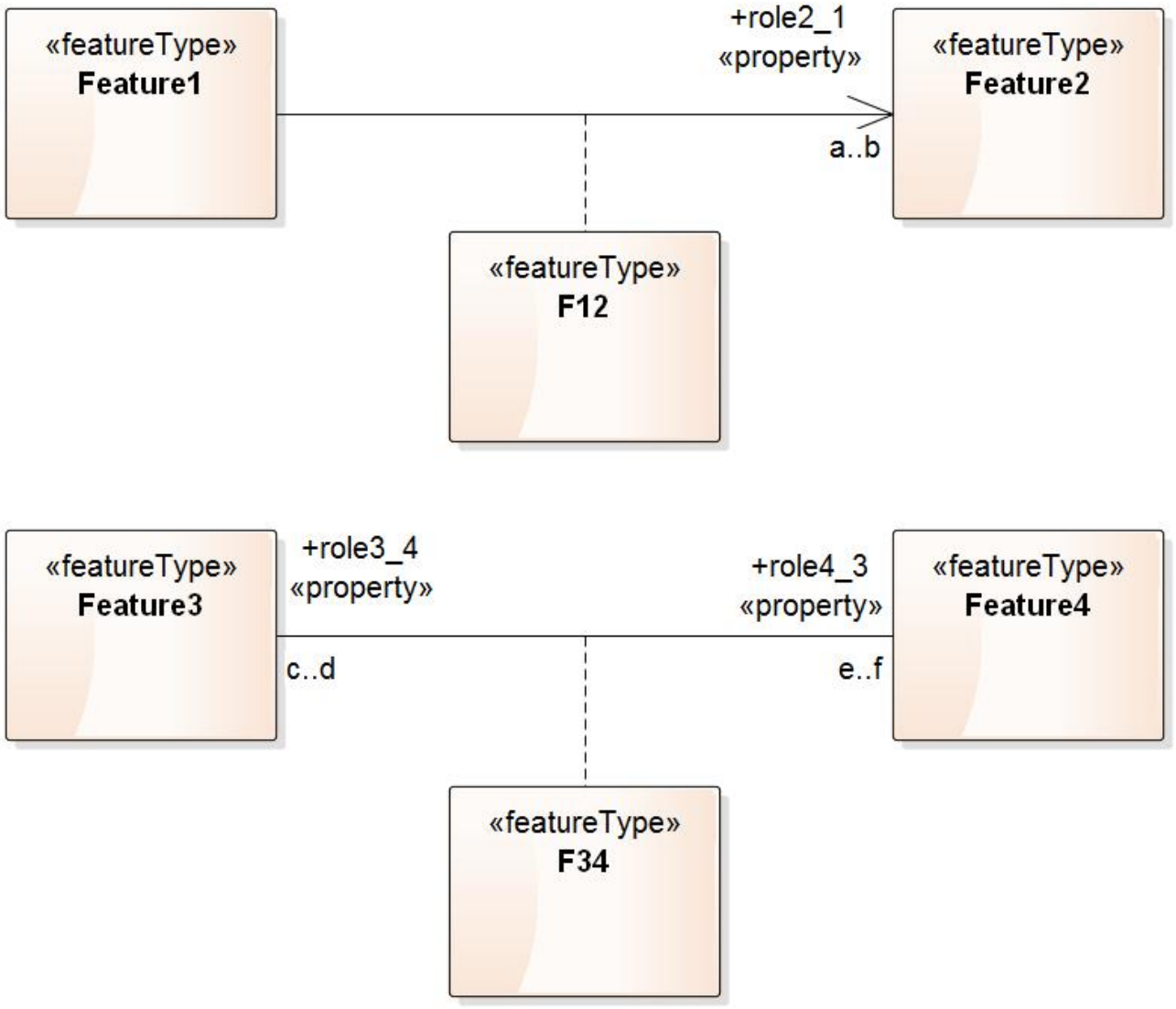

Figure 16. Model with association classes (source: OGC GML 3.3)

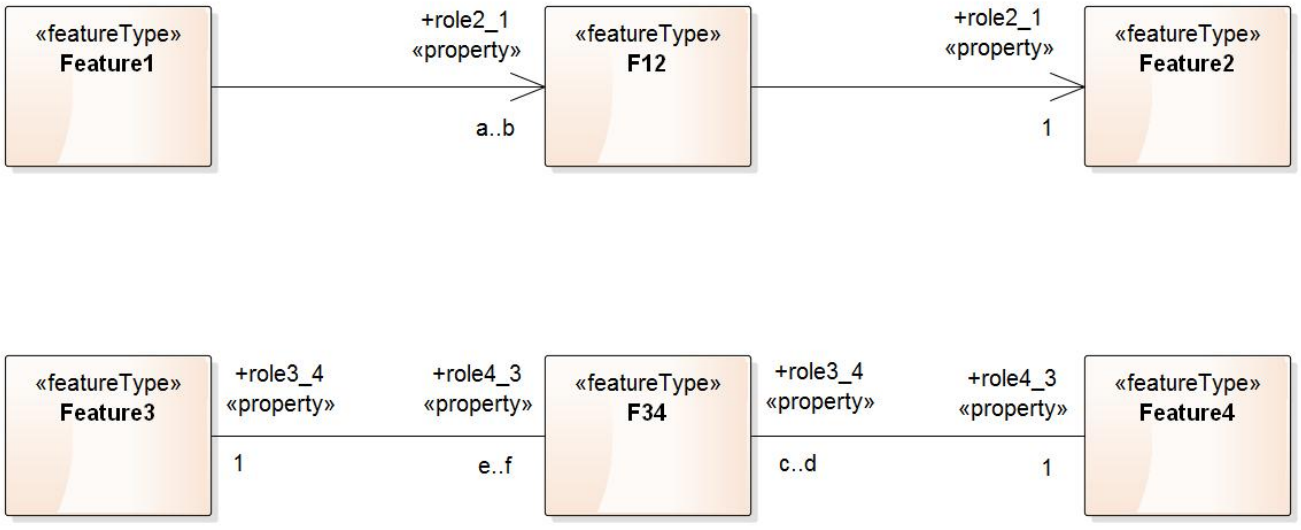

Figure 17. Association classes converted to intermediate classes (source: OGC GML 3.3)

The same approach should be applied when converting from UML to RDF/OWL. However, because this mapping may be relevant for other ShapeChange targets, it is realized as a ShapeChange transformation. The transformation will be performed before executing the OWL target. The OWL target - and other targets - then convert the resulting UML construct as usual (based upon the rules for converting classes, associations, and properties).

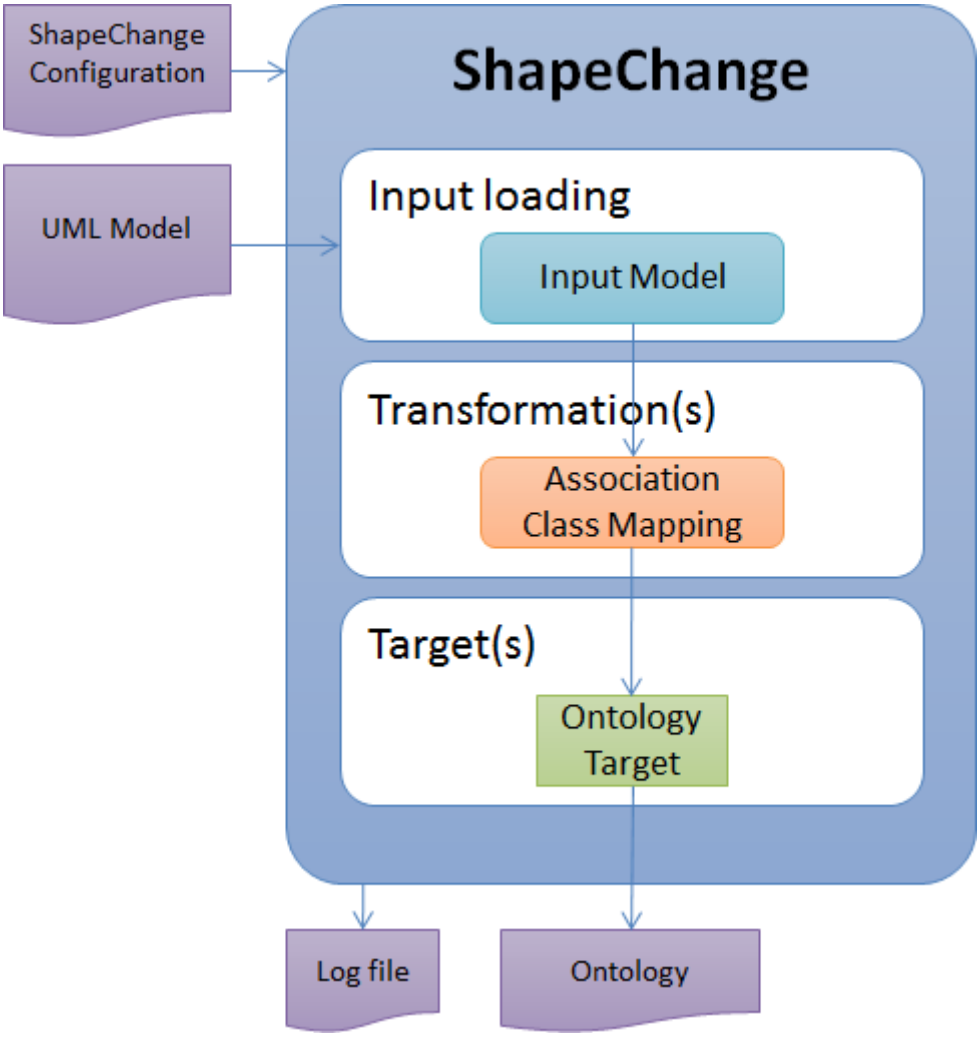

Figure 18. Ontology derivation workflow - with mapping of association classes

The following listing provides an example of a ShapeChange configuration element with the relevant transformation.

```
<Transformer input="A" id="B"
class="de.interactive_instruments.ShapeChange.Transformation.Flattening.AssociationCla
ssMapper"/>
```

NOTE	OCL constraints in the original schema might contain expressions that navigate to the association class or via the association that is (bound to) the association class. If the model is transformed according to the pattern from GML 3.3, then the paths in these OCL expressions would need to be updated. OCL expressions that navigate from the association class should not be affected. Whether or not this is significant depends on: a) if the schema contains OCL expressions that would need to be updated and b) if the target implementation would implement the OCL expression in some way. Regarding condition (b): The XML Schema Target supports derivation of Schematron constraints from OCL constraints. There, an update of the OCL expression when transforming association classes before executing the XML Schema Target would be necessary for the derivation of Schematron constraints. In the UML-to-RDF conversion, OCL expressions may be documented, but actual parsing and implementation - for example as OWL property restrictions - is not performed yet. See the future work section for further details.

8.2.6. Constraints

UML classes and properties can have constraints that express additional requirements. For example, the value range of an integer-valued property can be restricted to the interval [0:100], or the existence of values for a specific property can be prohibited in the scope of a certain class. Constraints add precision to models.

ISO 19103 recommends the use of the Object Constraint Language (OCL) (v2) for expressing constraints. Accordingly, application schemas often contain constraints written in OCL. In some cases, other types of constraints are used as well. A simple human-readable text, for example, can express modeling intent to a human reader. Natural-language-like sentences expressed using the Semantics of Business Vocabulary and Business Rules (SBVR) can be read by humans and - depending on the grammar used - also by machines (see [2] for further details).

This section documents the conversion of constraints contained in the UML model to RDF/OWL. At first, background information is provided about the current state of: a) constraint conversion in ISO 19150-2, b) constraint usage in the NAS, and c) support of constraints in ShapeChange. Then, the mapping from NAS constraints to the ShapeChange constraint model is discussed. Finally, the rules for converting constraints to RDF/OWL are specified.

Background: ISO 19150-2

According to ISO 19150-2, clause 6.11, a constraint is represented using the *iso19150-2:constraint* annotation property, which is defined in the base ontology as a string-valued property. Nothing specific is said about how the string value of this annotation shall be constructed. ISO 19150-2 recognizes that OWL has some mechanisms for describing constraints. However, “these constraints are somewhat different to or have a different purpose of UML CONSTRAINTs” (ISO 19150-2, section 6.11.2).

The base ontology is described in ISO 19150-2, Annex D. The ontology name is http://def.isotc211.org/iso19150/-2/2012/base, and its namespace is http://def.isotc211.org/iso19150/-2/2012/base# with the prefix iso19150-2. The iso19150-2:constraint property is defined as follows:

```
<owl:AnnotationProperty rdf:about="&iso19150-2;constraint">
  <rdfs:domain rdf:resource="&owl;Class"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:AnnotationProperty>
```

NOTE	Because the annotation property domain is set to owl:Class, the iso19150-2:constraint can only be used on resources that are represented as OWL classes.

Background: NAS OCL Constraints

NAS constraints are usually expressed in OCL. They are documented in, for example, the NAS v7.0 workbook, Entities tab, Col. M. They can also be found online in the NSG Registry.

In the NAS entry for an OCL constraint, the formal OCL expression is the “Definition” of the constraint, while the English statement is captured as the “Description”.

NAS constraints have their own:

- Code (e.g., “ReligiousSignificanceExc”)
- Name (e.g., “Religious Significance – Excluded")
- Definition (e.g., inv: religiousSignificance→isEmpty())
- Description (e.g., "An access zone never has religious significance (which is otherwise inherited from [FeatureEntity])")
- Status (e.g., Valid)
- Date (e.g., 11 Jul 2013)

Background: Constraints in ShapeChange

ShapeChange supports constraints on classes and properties. Constraints can be expressed in OCL, as plain text, and as a profile of SBVR. Constraints expressed with OCL and SBVR can be parsed into an internal language to derive additional artifacts, such as Schematron files for validating XML data.

ShapeChange supports the following constraint properties:

Table 9. ShapeChange Constraint Properties

Constraint property name	Description	Supported in text constraint	Supported in OCL constraint	Supported in constraint parsed from SBVR
Name	Identifies the constraint.	Yes	Yes	Yes
Text	The textual representation of the constraint.	Yes	Yes	Yes

Constraint property name	Description	Supported in text constraint	Supported in OCL constraint	Supported in constraint parsed from SBVR
Status	Expresses some state of refinedness, validity or purpose of the constraint; NOTE: This property is experimental.	Yes	Yes	Yes
Context model element	References the class or property on which the constraint is defined.	Yes	Yes	Yes
Comment	Human readable comment. NOTE: Comments can be parsed from the constraint text. In the case of OCL comments, they would be included within '/*' and '*/'. In the case of comments parsed from SBVR, they represent the original text.	No	Yes	Yes

Mapping NAS Constraints to ShapeChange Constraints

The information items defined for NAS constraints (code, name, etc.) need to be mapped to the internal model of ShapeChange. The following table describes a possible mapping:

NAS OCL constraint property	Mapping to ShapeChange OCL constraint property
Code	*No direct mapping (but could be provided as part of the name)*
Name	Name
Definition	Text
Description	Comment(s) (and, optionally, included in the text - i.e. the OCL expression - within '/*' and '*/')
Status	Status
Date	*No direct mapping*

The actual mapping depends on how the model is loaded. If the NAS model is encoded in an Enterprise Architect repository (e.g., an .eap file), then the NAS constraint properties will already have been mapped to the EA constraint model. If the model is encoded in a NAS-conformant database, then all NAS constraint properties are directly accessible and available for a mapping.

Conversion of Constraints to RDF/OWL

The conversion of constraints as specified by ISO 19150-2 is a starting point. There are two issues:

- ISO 19150-2 does not provide a precise requirement for constructing the string value of the *iso19150-2:constraint* annotation property. However, in this case it is understandable why no

specific rules are provided by ISO 19150-2:

  - Constraints can be expressed in a number of ways: OCL, natural language, based on SBVR, etc. For each type of constraint, a different set of information items may be available and relevant (name, text, comment(s), etc.).
  - Communities can have different preferences as to how the information items available for a specific type of constraint are encoded in RDF (e.g., name and comments, or alternatively, just the text).

- The domain of the *iso19150-2:constraint* annotation property is owl:Class. Thus, adding the constraint annotation to properties would not be allowed. There are two reasons why this might not be an issue:
  - Most often, constraints are defined on classes, not properties. If an application schema does not declare constraints on properties, then the domain of the iso19150-2:constraint annotation is suitable.
  - It is not entirely clear if software that processes an ontology would report an error if the iso19150-2:constraint annotation was added to properties. A test with Protégé ignored the use of an annotation property on resources other than the one declared as the domain of the annotation property (a test with object properties assigned to the wrong class - also declaring that class as disjoint with the class that was defined as the domain of the property - resulted in an error). So software might ignore the annotation property domain.

Two conversion rules have been defined to specify how constraints can be encoded.

Human Readable Text Only

If *rule-owl-all-constraints-humanReadableTextOnly* is enabled, then only the human readable text of a constraint is encoded. This means:

- For ShapeChange text constraints as well as constraints parsed from SBVR, the complete text is encoded.
- For ShapeChange OCL constraints, the values of the comment property are encoded (multiple comments are concatenated using a single space character as separator).
- In both cases, the constraint name is prepended to the text.

By Constraint Mapping

If *rule-owl-all-constraints-byConstraintMapping* is enabled, then constraints on UML classes and properties are converted to an RDF property, as specified via the *constraintMappings* configuration (using defaults documented above as fallback).

With *constraintMappings*, the mappings to a string value can be explicitly defined for each constraint type. Templates are used to specify how the properties of a given constraint are mapped to a string value. This is similar to how *descriptorTargets* work. A constraint mapping also allows choosing an RDF property other than iso19150-2:constraint for representing constraints.

The following table documents the structure of a *constraint mapping*. An example is also provided. The XML Schema can be found in the Annex on XML Schema Documents.

Table 10. Constraint Mapping

Information Item (configured via XML attribute)	**Datatype & Structure**	**Required / Optional**	**Default Value**	**Description**
constraintType	enum: "Text", "OCL", or "FOL"	Required	*not applicable*	Identifies the type of constraint for which the mapping is defined.
target	string; the syntax shall follow QNames, with the prefix being equal to the namespace abbreviation of a namespace that is contained in the configuration of the ShapeChange ontology target	Optional	iso19150-2:constraint	IRI of an RDF property or OWL annotation property that will be used to represent the constraint; the subject is the OWL class or property representing the UML class or property on which the constraint is defined (i.e., its context model element), and the object is a language tagged string. The string content is determined by the template. The configuration parameter "language" (in the ontology target configuration) provides the value of the language tag.
template	string	Required	*not applicable*	Text template where an occurrence of the field "[[*constraint property ID*]]" is replaced with the value of that property. The available constraint property IDs as well as additional details are documented in the table below.
noValue	string	Optional	*the empty string*	If a constraint property used in a template has no value, then this information item provides the text to use instead (e.g., "N/A" or "FIXME").
multiValueConnectorToken	string	Optional	*a single space character*	If a constraint property used in a template has multiple values, they are concatenated to a single string value using this token as connector between two values.
format	enum: *string*, or *langString*	Optional	*string*	Defines the format of the target property value: * *langString*: language-tagged string; the configuration parameter "language" (in the ontology target configuration) provides the value of the language tag * *string*: string without language tag

NOTE

Only a single mapping is allowed per constraint type.

A constraint is mapped using the template that is most specific for the given constraint type. For example, FOL constraints are subtypes of Text constraints. If the configuration contains mappings for both Text and FOL constraints, then the latter is applied for a FOL constraint, but not the former. If the configuration only contained a mapping for Text constraints, then it would be applied for a FOL constraint (because no mapping is provided that is more specific for a FOL constraint).

If the ShapeChange configuration does not contain a mapping for a constraint type encountered in the application schema (or one of its supertypes), ShapeChange will report a warning and use the default values for *noValue* and *multiValueConnectorToken*, as well as the default template: [[name]]: [[text]]

The following example shows the use of constraintMappings.

```
<sc:constraintMappings>
  <sc:ConstraintMapping constraintType="Text" template="[[name]]: [[text]]"
noValue="FIXME"/>
  <sc:ConstraintMapping constraintType="OCL"
    template="[[name]]: [[comment]] --- OCL expression: [[text]]"/>
</sc:constraintMappings>
```

The following table documents which properties are available for which type of constraint supported by ShapeChange.

Table 11. ShapeChange constraint property IDs and further details

Constraint Property ID	Single- or multi-valued (S/M)	Available in Text Constraint (Y/N)	Available in OCL Constraint (Y/N)	Available in FOL Constraint (Y/N)
name	S	Y	Y	Y
text	S	Y	Y	Y
status	S	Y	Y	Y
comment	M	N	Y	Y

NOTE

The context model element cannot be used in the template of a constraint mapping. The element is represented by the OWL class or property to which the target annotation property (as defined by the constraint mapping) is added.

Rule Execution Priority and Dependencies

The conversion rules for constraints are mutually exclusive. If both rules are enabled, ShapeChange will log a warning and continue processing as if only *rule-owl-all-constraints-humanReadableTextOnly* was enabled. If none of these rules is enabled, constraints will not be encoded.

8.3. Implementation

All of the UML to RDF/OWL/SKOS conversion rules and parameters documented in this chapter have been implemented.

The implementation is based on Apache Jena (http://jena.apache.org/).

The output of ShapeChange can be written in the formats supported by Apache Jena:

Table 12. Formats supported for writing an ontology

Format	value of "outputFormat" configuration parameter
Turtle	turtle
RDF/XML	rdfxml
N-Triples	ntriples
JSON-LD	jsonld
RDF/JSON	rdfjson
TriG	trig
N-Quads	nquads
TriX	trix
RDF Thrift	rdfthrift

For further details, see https://jena.apache.org/documentation/io/#formats

8.4. NAS Ontology Encoding Rule

The following sections contain extracts of ShapeChange configurations with which an ontology as well as a taxonomy were derived from the NSG Application Schema.

NOTE These extracts only serve as examples. They do not necessarily reflect the configuration that was used to generate any of the publicly available NSG Enterprise Ontology (NEO) or NSG Taxonomy (NTAX) documents.

8.4.1. Deriving the NSG Enterprise Ontology (NEO)

```
<?xml version="1.0" encoding="UTF-8"?>
<TargetOwl inputs="TRF2"
class="de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150"
 mode="enabled">
 <targetParameter name="outputDirectory" value="testResults/NEO_v8.0"/>
 <targetParameter name="outputFormat" value="rdfxml"/>
 <targetParameter name="defaultEncodingRule" value="NEO"/>
 <targetParameter name="language" value="en"/>
```

```
 <targetParameter name="defaultTypeImplementation" value="owl:Class"/>
 <targetParameter name="ontologyNameCode" value="neo"/>
 <targetParameter name="source" value="NSG Enterprise Ontology (NEO) Standard (2016-
mm-dd)"/>
 <targetParameter name="URIbase" value="http://api.nsgreg.nga.mil/ontology"/>
 <targetParameter name="skosConceptSchemeSuffix" value="_ConceptScheme"/>
 <targetParameter name="codeNamespaceForEnumerations"
  value="http://api.nsgreg.nga.mil/ontology/neo-enum/8.0"/>
 <targetParameter name="codeListOwlClassNamespaceForEnumerations"
  value="http://api.nsgreg.nga.mil/ontology/neo-enum/8.0"/>
 <descriptorTargets>
  <DescriptorTarget appliesTo="ontology" target="rdfs:label"
template="[[TV:ontologyName]]"
   format="langString"/>
  <DescriptorTarget appliesTo="ontology" target="skos:definition"
   template="![CDATA[Definition: [[TV:ontologyDefinition]]  Description:
[[TV:ontologyDescription]]]]"
   noValueText="[None Specified]" format="langString"/>
  <DescriptorTarget appliesTo="ontology" target="rdfs:isDefinedBy"
   template="[[TV:ontologyResourceURI]]" format="IRI"/>
  <DescriptorTarget appliesTo="ontology" target="skos:prefLabel"
template="[[TV:ontologyName]]"
   format="langString"/>
  <DescriptorTarget appliesTo="ontology" target="skos:altLabel"
template="[[TV(|):aliasList]]"
   multiValueBehavior="splitToMultipleTargets" format="langString"/>
  <DescriptorTarget appliesTo="class" target="rdfs:label"
template="[[TV:primaryCode]]"
   format="langString"/>
  <DescriptorTarget appliesTo="class" target="skos:definition"
   template="![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
   noValueText="[None Specified]" format="langString"/>
  <DescriptorTarget appliesTo="class" target="rdfs:isDefinedBy"
   template="http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
  <DescriptorTarget appliesTo="class" target="skos:prefLabel" template="[[TV:name]]"
   format="langString"/>
  <DescriptorTarget appliesTo="class" target="skos:altLabel"
template="[[TV(|):aliasList]]"
   multiValueBehavior="splitToMultipleTargets" format="langString"/>
  <DescriptorTarget appliesTo="property" target="rdfs:label"
template="[[TV:primaryCode]]"
   format="langString"/>
  <DescriptorTarget appliesTo="property" target="skos:definition"
   template="![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
   noValueText="[None Specified]" format="langString"/>
  <DescriptorTarget appliesTo="property" target="rdfs:isDefinedBy"
   template="http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
  <DescriptorTarget appliesTo="property" target="skos:prefLabel"
template="[[TV:name]]"
   format="langString"/>
  <DescriptorTarget appliesTo="property" target="skos:altLabel"
```

```
template="[[TV(|):aliasList]]"
   multiValueBehavior="splitToMultipleTargets" format="langString"/>
  <DescriptorTarget appliesTo="conceptscheme" target="rdfs:label"
   template="[[TV:primaryCode]]_ConceptScheme" format="langString"/>
  <DescriptorTarget appliesTo="conceptscheme" target="skos:definition"
   template="![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
   noValueText="[None Specified]" format="langString"/>
  <DescriptorTarget appliesTo="conceptscheme" target="rdfs:isDefinedBy"
   template="http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
  <DescriptorTarget appliesTo="conceptscheme" target="skos:prefLabel"
   template="[[TV:name]] - Concept Scheme" format="langString"/>
 </descriptorTargets>
 <xi:include href="test/config/StandardMapEntries-owl.xml"/>
 <rdfMapEntries>
  <RdfTypeMapEntry type="SecurityAttributesGroupType" target="ntk:RequiresType"/>
  <RdfTypeMapEntry type="ISM_Notice" target="icism:NoticeType"/>
  <RdfTypeMapEntry type="NTKAccess" target="ntk:RequiresType"/>
  <RdfTypeMapEntry type="RevisionRecall" target="rr:RevisionRecallType"/>
 </rdfMapEntries>
 <constraintMappings>
  <ConstraintMapping constraintType="OCL" template="[[name]]: [[comment]]  OCL
expression: [[text]]"
   noValue="[None Specified]" format="langString"/>
 </constraintMappings>
 <xi:include href="test/config/StandardNamespaces-owl.xml"/>
 <namespaces>
  <Namespace nsabr="icism"
ns="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public#"
   location="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public"/>
  <Namespace nsabr="ntk" ns="https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-
V10-Public#"
   location="https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-V10-Public"/>
  <Namespace nsabr="rr"
ns="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public#"
   location="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public"/>
 </namespaces>
 <rules>
  <EncodingRule name="NEO" extends="*">
   <rule name="rule-owl-pkg-singleOntologyPerSchema"/>
   <rule name="rule-owl-pkg-ontologyName-code"/>
   <rule name="rule-owl-pkg-ontologyName-appendVersion"/>
   <rule name="rule-owl-pkg-versionInfo"/>
   <rule name="rule-owl-pkg-versionIRI"/>
   <rule name="rule-owl-pkg-versionIRI-avoid-duplicate-version"/>
   <rule name="rule-owl-pkg-dctSourceTitle"/>
   <rule name="rule-owl-cls-iso191502IsAbstract"/>
   <rule name="rule-owl-cls-generalization"/>
   <rule name="rule-owl-cls-disjoint-classes"/>
   <rule name="rule-owl-cls-encode-featuretypes"/>
   <rule name="rule-owl-cls-encode-objecttypes"/>
   <rule name="rule-owl-cls-encode-mixintypes"/>
```

```
    <rule name="rule-owl-cls-encode-datatypes"/>
    <rule name="rule-owl-cls-encode-basictypes"/>
    <rule name="rule-owl-prop-general"/>
    <rule name="rule-owl-prop-range-global"/>
    <rule name="rule-owl-prop-localScopeAll"/>
    <rule name="rule-owl-prop-multiplicityAsUnqualifiedCardinalityRestriction"/>
    <rule name="rule-owl-prop-iso191502AssociationName"/>
    <rule name="rule-owl-prop-inverseOf"/>
    <rule name="rule-owl-prop-iso191502Aggregation"/>
    <rule name="rule-owl-all-constraints-byConstraintMapping"/>
    <rule name="rule-owl-cls-union"/>
    <rule name="rule-owl-cls-unionSets"/>
    <rule name="rule-owl-cls-enumerationAsCodelist"/>
    <rule name="rule-owl-cls-codelist-external"/>
    <rule name="rule-owl-cls-codelist-19150-2"/>
    <rule name="rule-owl-cls-codelist-19150-2-objectOneOfForEnumeration"/>
    <rule name="rule-owl-cls-codelist-19150-2-differentIndividuals"/>
    <rule name="rule-owl-cls-codelist-19150-2-owlClassInDifferentNamespace"/>
    <rule name="rule-owl-prop-code-broader-byBroaderListedValue"/>
  </EncodingRule>
 </rules>
</TargetOwl>
```

8.4.2. Deriving the NSG Taxonomy (NTAX)

```
<?xml version="1.0" encoding="UTF-8"?>
<TargetOwl inputs="TRF3"
class="de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150"
 mode="enabled">
 <targetParameter name="outputDirectory" value="testResults/NTAX_v8.0"/>
 <targetParameter name="outputFormat" value="rdfxml"/>
 <targetParameter name="defaultEncodingRule" value="NEO"/>
 <targetParameter name="language" value="en"/>
 <targetParameter name="defaultTypeImplementation" value="owl:Class"/>
 <targetParameter name="ontologyNameCode" value="ntax"/>
 <targetParameter name="source" value="NSG Taxonomy (NTAX) Standard (draft)"/>
 <targetParameter name="URIbase" value="http://api.nsgreg.nga.mil/taxonomy"/>
 <descriptorTargets>
  <DescriptorTarget appliesTo="ontology" target="rdfs:label"
template="[[TV:taxonomyName]]"
   format="langString"/>
  <DescriptorTarget appliesTo="ontology" target="skos:definition"
   template="![CDATA[Definition: [[TV:taxonomyDefinition]]  Description:
[[TV:taxonomyDescription]]]]"
   noValueText="[None Specified]" format="langString"/>
  <DescriptorTarget appliesTo="ontology" target="rdfs:isDefinedBy"
   template="[[TV:taxonomyResourceURI]]" format="IRI"/>
  <DescriptorTarget appliesTo="ontology" target="skos:prefLabel"
template="[[TV:taxonomyName]]"
   format="langString"/>
```

```
  <DescriptorTarget appliesTo="ontology" target="skos:altLabel"
template="[[TV(|):aliasList]]"
   multiValueBehavior="splitToMultipleTargets" format="langString"/>
  <DescriptorTarget appliesTo="class" target="rdfs:label"
template="[[TV:primaryCode]]"
   format="langString"/>
  <DescriptorTarget appliesTo="class" target="skos:definition"
   template="![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
   noValueText="[None Specified]" format="langString"/>
  <DescriptorTarget appliesTo="class" target="rdfs:isDefinedBy"
   template="http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
  <DescriptorTarget appliesTo="class" target="skos:prefLabel" template="[[TV:name]]"
   format="langString"/>
  <DescriptorTarget appliesTo="class" target="skos:altLabel"
template="[[TV(|):aliasList]]"
   multiValueBehavior="splitToMultipleTargets" format="langString"/>
 </descriptorTargets>
 <xi:include href="test/config/StandardMapEntries-owl.xml"/>
 <rdfMapEntries>
  <RdfTypeMapEntry type="Binary" target="owl:Class"/>
  <RdfTypeMapEntry type="SecurityAttributesGroupType" target="ntk:RequiresType"/>
  <RdfTypeMapEntry type="ISM_Notice" target="icism:NoticeType"/>
  <RdfTypeMapEntry type="NTKAccess" target="ntk:RequiresType"/>
  <RdfTypeMapEntry type="RevisionRecall" target="rr:RevisionRecallType"/>
 </rdfMapEntries>
 <rdfConversionParameters>
  <TypeConversionParameter type="Dataset" schema="NSG Application Schema"
   subClassOf="ntax:RecordSet"/>
  <TypeConversionParameter type="Series" schema="NSG Application Schema"
subClassOf="ntax:RecordSet"/>
  <TypeConversionParameter type="DataQuality" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="DataIdentification" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="DigitalTransferOptions" schema="NSG Application
Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="DataProcessStep" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="ResourcePartyOrg" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="ResourceConstraints" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="LegalConstraints" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="DataSource" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="DataLineage" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="DispositionHold" schema="NSG Application Schema"
   subClassOf="ntax:RecordMetadata"/>
```

```
  <TypeConversionParameter type="ElectronicRecordsManagement" schema="NSG Application
Schema"
    subClassOf="ntax:RecordMetadata"/>
  <TypeConversionParameter type="RecordDisposition" schema="NSG Application Schema"
    subClassOf="ntax:RecordMetadata"/>
 </rdfConversionParameters>
 <constraintMappings>
  <ConstraintMapping constraintType="OCL" template="[[name]]: [[comment]]  OCL
expression: [[text]]"
    noValue="[None Specified]" format="langString"/>
 </constraintMappings>
 <xi:include href="test/config/StandardNamespaces-owl.xml"/>
 <namespaces>
  <Namespace nsabr="ntax" ns="http://api.nsgreg.nga.mil/taxonomy/ntax/8.0#"/>
  <Namespace nsabr="icism"
ns="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public#"
    location="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public"/>
  <Namespace nsabr="ntk" ns="https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-
V10-Public#"
    location="https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-V10-Public"/>
  <Namespace nsabr="rr"
ns="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public#"
    location="https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public"/>
 </namespaces>
 <rules>
  <EncodingRule name="NEO" extends="*">
   <rule name="rule-owl-pkg-singleOntologyPerSchema"/>
   <rule name="rule-owl-pkg-ontologyName-code"/>
   <rule name="rule-owl-pkg-ontologyName-appendVersion"/>
   <rule name="rule-owl-pkg-versionInfo"/>
   <rule name="rule-owl-pkg-versionIRI"/>
   <rule name="rule-owl-pkg-versionIRI-avoid-duplicate-version"/>
   <rule name="rule-owl-pkg-dctSourceTitle"/>
   <rule name="rule-owl-cls-iso191502IsAbstract"/>
   <rule name="rule-owl-cls-generalization"/>
   <rule name="rule-owl-cls-disjoint-classes"/>
   <rule name="rule-owl-cls-encode-featuretypes"/>
   <rule name="rule-owl-cls-encode-objecttypes"/>
   <rule name="rule-owl-cls-encode-mixintypes"/>
  </EncodingRule>
 </rules>
</TargetOwl>
```

Annex A: Comparison of Encoding Rules in ISO 19150-2 Draft and Final

This appendix documents the results of an analysis that compared the requirements from ISO 19150-2 DIS with the requirements from ISO 19150-2 IS. It also documents the changes, extensions and limitations of the ShapeChange ontology target that was developed based upon ISO 19150-2 DIS, in comparison to that specification (i.e. ISO 19150-2 DIS).

NOTE	Due to the revision of the ShapeChange ontology target (based on ISO 19150-2) in OGC Testbed 12, the parameters and conversion rules mentioned in this annex are mostly outdated (since they apply to the previous version of the ShapeChange ontology target).

Table 13. Rules for mapping ISO geographic information UML models to OWL ontologies

ISO 19150-2 DIS requirement & chapter	**ISO 19150-2 IS requirement, chapter & changes**	**ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS**
19150-2owl:ontologyName (chapter 6.2.2 Ontology Name)	19150-2package:ontologyName (chapter 6.2.2 Ontology Name) CHANGE(S): umlPackageName shall be in lower case (instead of UpperCamelCase)	Extension: If the conversion rule “rule-owl-pkg-app-schema-code” is active, the code for the application schema is used (tagged value “xmlns” specified by GML 3.2 / ISO 19136) instead of umlPackageName Extension: sub-packages of an application schema package do not become separate ontologies, but are part of the application schema ontology, if conversion rule “rule-owl-pkg-singleOntologyPerSchema” is active Note: The URIbase is set using the parameter of the same name
19150-2owl:rdfNamespace (chapter 6.2.3 RDF namespace for ontology)	19150-2package:rdfNamespace (chapter 6.2.3 RDF namespace for ontology) NO CHANGES	-
19150-2owl:className (6.2.4 Class name)	19150-2package:className (6.2.4 Class name) NO CHANGES	-

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:datatypeName (6.2.5 Datatype name)	19150-2package:datatypeName (6.2.5 Datatype name) NO CHANGES	-
19150-2owl:propertyName (6.2.6 Property name)	19150-2package:propertyName (6.2.6 Property name) No change, but addition of the following note: “In addition to its propertyLocalName, a propertyName can optionally include its umlClassName when the property is scoped to the class or if the same propertyLocalName may be used by another property of another class with a different semantics.”	Change: The properties that become global properties are not automatically determined by the uniqueness of the property name, but explicitly controlled using the parameter “globalProperties”. If “*” is used, all properties will be global. In case of multiple properties with the same name and conflicting annotations or ranges, the conflicts need to be resolved manually. NOTE: if a property is not marked to be global then it is automatically scoped to its class. Currently there is no automatic detection of uniqueness [of a property name] within a package or the whole schema. NOTE: if a property with the same name has already been encoded, then the differences are logged and the already existing one is used, rather than the new/current one.
19150-2owl:codeName (6.2.7 Names for codelists and their members)	19150-2package:codeName (6.2.7 Names for codelists and their members) ADDITION: collectionName = codeNamespace className "Collection"	Extension: if the conversion rule “rule-owl-cls-codelist-external” is active, code lists are not converted. See also the comments on sub-clause 6.8.2.

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:package (6.3 Packages)	19150-2package:package (6.3 Packages) Clarification that the UML model of a geographic information standard shall correspond to ontologies rather than packages with stereotype <<requirementsClass>> Some rewording regarding the import of the base ontology from ISO 19150-2	Change: dc:source is a text (specified by parameter “source”) as the rule in ISO/DIS 19150-2 is based on the assumption that the package is from the harmonized model, i.e. defined in a standard of ISO/TC 211. Change: owl:versionInfo uses the version information in tagged value “version” instead of a date.
19150-2owl:class (6.4 Class)	19150-2package:class (6.4 Class) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source	Extension: if conversion rule “rule-owl-all-suppress-dc-source” is active, dc:source is omitted (as the source statement in the ontology applies, too) Change: skos:notation is used for the class name, skos:prefLabel is used for an (human-readable) alias, if provided Change: skos:scopeNote is provided for a description, if provided
19150-2owl:abstractClass (6.5 Abstract class)	19150-2package:abstractClass (6.5 Abstract class) NO CHANGES	-

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:stereotype (6.6. Class stereotype)	REMOVED – and thus the requirement to inherit from classes representing stereotypes has been removed in all relevant clauses, too	Change: ISO/DIS 19150-2 includes stereotypes and tagged values in the ontology. In general, stereotypes and tagged values are UML-specific extension mechanisms. They should only be supported in schema conversion rules that map the values to naive RDFS/OWL constructs and carry relevant information. For most tags there is little or no apparent value. For example, there is no value in representing tagged values supporting the GML schema conversion rules in the ontology. Therefore, stereotypes and tagged values are in general suppressed.

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:attribute-dataProperty (6.7.3.1 OWL data property rules) & 19150-2owl:attribute-objectProperty (6.7.3.2)	19150-2package:attribute-dataProperty (6.6.3.1 OWL data property rules) & 19150-2package:attribute-objectProperty (6.6.3.2) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source	*The changes and extensions listed for "Class" apply here, too.* Limitation: gco:Datatypes are not supported Change: rdfs:domain is not provided for global properties Extension: rdfs:range supports owl:unionOf for cases where multiple UML attributes are "merged" to a single, global RDF property, see comments on sub-clause 6.2.6. Extension: Attributes may be implemented using other RDF vocabularies or suppressed (using MapEntry elements in the configuration with a param="property" or "propertyByValueType"). Extension: For ranges specified by types from the ISO Harmonized Model or other imported schemas, implementations may be specified in MapEntry elements in the configuration (param="datatype").
19150-2owl:enumeration (6.8.2 Enumeration)	19150-2package:enumeration (6.7.1 Enumeration) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source	*The changes and extensions listed for "Class" apply here, too.*

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:codelist & 19150-2owl:codelistextension (6.8.2 Code list)	19150-2package:codelist & 19150-2package:codelistextension (6.7.2 Code list) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source	*The changes and extensions listed for "Class" apply here, too.* Extension: If the conversion rule "rule-owl-cls-codelist-external" is active, code lists are not converted. This should be the normal case as usually code lists are managed outside of the application schemas. If no tagged value "codelist" or "vocabulary" is present, rdfs:Resource is used as the range, otherwise the resource identified by the URI.
19150-2owl:union (6.9 Union)	19150-2package:union (6.8 Union) NO CHANGES	Limitation: Unions are represented by stub classes as the schema conversion rule for union data types in ISO/DIS 19150-2 is insufficient. It does not handle common cases where values are a mix of object or data types or the same value type is used by more than one option.

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:multiplicity (6.10 Multiplicity)	19150-2package:multiplicity (6.9 Multiplicity) NO CHANGES	Extension: If the conversion rule “rule-owl-prop-suppress-cardinality-restrictions” is active, cardinality restrictions are not included. Extension: If the conversion rule “rule-owl-prop-suppress-allValuesFrom-restrictions” is active, range restrictions are not included in all cases when the value would be identical with the range information of the property. Extension: If the conversion rule “rule-owl-prop-voidable-as-minCardinality0" is active and a property is voidable, there will be no minimum cardinality restriction.
19150-2owl:relationship-generalization (6.11.1 Generalization/inheritance)	19150-2package:relationship-generalization (6.10.1 Generalization/inheritance) NO CHANGES	-
19150-2owl:relationship-association (6.11.2 Association)	19150-2package:relationship-association (6.10.2 Association) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source	*The changes and extensions listed for “Class” and “Attribute” apply here, too.* Extension: If conversion rule “rule-owl-prop-suppress-asociation-names” is active, iso19150-2:associationName is not included. Limitation: ISO/DIS 19150-2 does not provide rules for association classes.
19150-2owl:relationship-aggregation (6.11.3 Aggregation)	19150-2package:relationship-aggregation (6.10.3 Aggregation) NO CHANGES	-

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2owl:constraint (6.12 Constraint)	19150-2package:constraint (6.11 Constraint) NO CHANGES (just a minor clarification that OWL supports constraints but not those we are looking for)	Extension: Constraints are only added if conversion rule “rule-owl-all-constraints” is active. Including OCL in an ontology is questionable. Probably the most reasonable way would be to include only the documentation of a constraint.
19150-2owl:taggedvalue (6.13 Tagged value)	REMOVED	*The changes and extensions listed for “Stereotype” apply here, too.*

Table 14. Rules for formalizing an application schema in OWL

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2app:identification (7.2. Rules for identification)	19150-2app:identification (7.2. Rules for identification) NO CHANGES	-
19150-2app:documentation-ontology (7.3.1 Ontology documentation)	19150-2app:documentation-ontology (7.3.1 Ontology documentation) Change: rdfs:isDefinedBy instead of dc:source	-
19150-2app:documentation-ontologyComponent (7.3.2 Ontology component documentation)	19150-2app:documentation-ontologyComponent (7.3.2 Ontology component documentation) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source Removed: annotation properties iso19150-2:isEnumeration and –isCodelist no longer used	*The changes and extensions listed for “Class”, “Attribute” and “Association” apply here, too.*

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2app:integration (7.4 Rules for integration)	19150-2app:integration (7.4 Rules for integration) NO CHANGES	-
19150-2app:GF_FeatureType (7.5 Rules for GF_FeatureType)	19150-2app:featureType (7.5 Rules for FeatureType) Change: uses rdfs:label instead of skos:prefLabel Removed: not inheriting from iso19150-2:FeatureType because stereotypes have been removed completely	*The changes and extensions listed for "Class" apply here, too.* Extension: A sub-class predicate to gfm:AnyFeature is only added, if conversion rule "rule-owl-cls-19150-2-features" is active. Extension: If conversion rule "rule-owl-cls-geosparql-features" is active, a sub-class predicate to geo:Feature from GeoSPARQL is added.
19150-2app:attributeType (7.6.1.1 Rules for GF_AttributeType)	19150-2app:attributeType (7.6.1.1 Rules for AttributeType) Removed: representation of stereotypes [dpEstimated] and [opEstimated] (probably replaced by ValueAssignment – see last row of this table) Change: uses rdfs:label instead of skos:prefLabel and rdfs:isDefinedBy instead of dc:source	*The changes and extensions listed for "Attribute" apply here, too.*
NOT EXISTENT (due to update of ISO 19150-2 IS to new version of ISO 19109 (v2015)) (GF_ThematicAttributeType is mentioned at the end of 7.6.1.1)	19150-2app:thematicAttributeType (7.6.1.2 Rules for ThematicAttributeType) NEW	*the DIS did not contain a specific requirement for thematic attribute types*
NOT EXISTENT (due to update of ISO 19150-2 IS to new version of ISO 19109 (v2015))	19150-2app:coverageFunctionAttributeType (7.6.1.3 Rules for CoverageFunctionAttributeType) NEW	*the DIS did not contain a specific requirement for coverage function attribute types*

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2app:attribute-GF_LocationAttributeType (7.6.1.2 Rules for GF_LocationAttributeType)	19150-2app:locationAttributeType (7.6.1.4 Rules for LocationAttributeType) NO CHANGES	-
19150-2app:attribute-GF_SpatialAttributeType (7.6.1.3 Rules for GF_SpatialAttributeType)	19150-2app:spatialAttributeType (7.6.1.5 Rules for SpatialAttributeType) NO CHANGES	-
19150-2app:attribute-GF_TemporalAttributeType (7.6.1.4 Rules for GF_TemporalAttributeType)	19150-2app:temporalAttributeType (7.6.1.6 Rules for TemporalAttributeType) NO CHANGES	-
19150-2app:attribute-GF_MetadataAttributeType (7.6.1.5 Rules for GF_MetadataAttributeType)	19150-2app:metadataAttributeType (7.6.1.7 Rules for MetadataAttributeType) NO CHANGES	-
NOT EXISTENT (due to update of ISO 19150-2 IS to new version of ISO 19109 (v2015))	19150-2app:qualityAttributeType (7.6.1.8 Rules for QualityAttributeType) NEW	*the DIS did not contain a specific requirement for coverage function attribute types*
19150-2app:attribute-attributeOfAttribute (7.6.1.6 Rules for attribute of attribute)	19150-2app:attributeOfAttribute (7.6.1.9 Rules for attribute of attribute) NO CHANGES	-
19150-2app:GF_Operation (7.6.2 Rules for operation)	19150-2app:operation (7.6.2 Rules for operation) NO CHANGES	-
19150-2app:GF_AssociationRole (7.6.3 Rules for GF_AssociationRole)	19150-2app:featureAssociationRole (7.6.3 Rules for FeatureAssociationRole)	*The changes and extensions listed for “Attribute” apply here, too.*

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
19150-2app:GF_AssociationType (7.6.4 Rules for GF_AssociationType)	19150-2app:featureAssociationType (7.6.4 Rules for FeatureAssociationType) Changes: dropped requirement to specify the rdfs:domain for an association property.	*The changes and extensions listed for "Association" apply here, too.*
19150-2app:GF_AggregationType (7.8 Rules for GF_AggregationType) (due to update of ISO 19150-2 IS to new version of ISO 19109 (v2015) this clause has been split up into two: one for aggregation and one for composition)	19150-2app:featureAggregationType (7.8 Rules for FeatureAggregationType) & 19150-2app:featureCompositionType (7.9 Rules for FeatureCompositionType) NO CHANGES	-
NOT EXISTENT (due to update of ISO 19150-2 IS to new version of ISO 19109 (v2015))	19150-2app:spatialAssociationType (7.10 Rules for SpatialAssociationType) NEW	*This requirement shall be implemented as requirement 19150-2app:featureAssociationType. Therefore the changes and extensions listed for "Association" apply here, too.*
NOT EXISTENT (due to update of ISO 19150-2 IS to new version of ISO 19109 (v2015))	19150-2app:temporalAssociationType (7.11 Rules for TemporalAssociationType) NEW	*This requirement shall be implemented as requirement 19150-2app:featureAssociationType. Therefore the changes and extensions listed for "Association" apply here, too.*
19150-2app:GF_InheritanceRelation (7.9 Rules for GF_InheritanceRelation)	19150-2app:inheritanceRelation (7.12 Rules for InheritanceRelation) NO CHANGES	-
19150-2app:GF_Constraint (7.10 Rules for GF_Constraint)	19150-2app:constraint (7.13 Rules for constraints) NO CHANGES	*The changes and extensions listed for "Constraint" apply here, too.*

ISO 19150-2 DIS requirement & chapter	ISO 19150-2 IS requirement, chapter & changes	ShapeChange Target (based on ISO/DIS 19150-2) – Changes, extensions, limitations compared to ISO 19150-2 DIS
NOT EXISTENT (as a separate chapter, but the ISO DIS had a mechanism to implement the [estimated] stereotype for properties – see 19150-2app:attribute-GF_AttributeType in chapter 7.6.1.1)	19150-2app:valueAssignment (7.14 Rules for ValueAssignment) NEW (chapter – the mechanism to implement the [estimated] stereotype for properties is different to that specified by the DIS)	-

Annex B: XML Schema Documents

This annex contains XML Schema definitions for ShapeChange extensions specified in OGC Testbed 12. The latest version of the configuration is available online at http://shapechange.net and https://github.com/ShapeChange/ShapeChange.

B.1. ConstraintLoader XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
  targetNamespace="http://shapechange.net/constraintLoaderConfiguration/1.0"
version="1.0.0"
  xml:lang="en" xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:cl="http://shapechange.net/constraintLoaderConfiguration/1.0">
  <element name="Constraints">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded" name="constraint">
          <complexType>
            <sequence>
              <element ref="cl:Constraint"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
  <element name="ProfileIdentifier">
    <complexType>
      <sequence>
        <element name="name" type="string"/>
        <element minOccurs="0" name="versionIndicator" type="string"/>
      </sequence>
    </complexType>
  </element>
  <element name="Constraint" type="cl:ConstraintType">
    <unique name="profileNameInProfiles">
      <selector xpath="cl:profile"/>
      <field xpath="cl:ProfileIdentifier/cl:name"/>
    </unique>
  </element>
  <complexType name="ConstraintType">
    <sequence>
      <element minOccurs="0" name="constraintName" type="string"/>
      <element minOccurs="0" name="constraintType" type="string"/>
      <element name="constraintExpression" type="string"/>
      <element minOccurs="0" name="comment" type="string"/>
      <element name="schemaPackageName" type="string"/>
      <element name="contextElementName" type="string"/>
```

```
      <element default="Class" minOccurs="0" name="contextElementType">
        <simpleType>
          <restriction base="string">
            <enumeration value="Class"/>
            <enumeration value="Property"/>
          </restriction>
        </simpleType>
      </element>
      <element maxOccurs="unbounded" minOccurs="0" name="profile">
        <complexType>
          <sequence>
            <element ref="cl:ProfileIdentifier"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

B.2. ProfileLoader XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
targetNamespace="http://shapechange.net/profileLoader/1.0"
  version="1.0.0" xml:lang="en" xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:pl="http://shapechange.net/profileLoader/1.0">
  <element name="ProfileInformation">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded" name="schema">
          <complexType>
            <sequence>
              <element maxOccurs="1" name="Schema">
                <complexType>
                  <sequence>
                    <element name="packageName" type="string"/>
                    <element maxOccurs="unbounded" name="requirement">
                      <complexType>
                        <sequence>
                          <element ref="pl:Requirement"/>
                        </sequence>
                      </complexType>
                    </element>
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
```

```
      </sequence>
    </complexType>
    <unique name="packageNameOfSchemaInSchemas">
      <selector xpath="pl:schema"/>
      <field xpath="pl:Schema/pl:packageName"/>
    </unique>
  </element>
  <element abstract="false" name="Requirement" type="pl:RequirementType">
    <unique name="profileNameInProfiles">
      <selector xpath="pl:profile"/>
      <field xpath="pl:Profile/pl:identifier/pl:ProfileIdentifier/pl:name"/>
    </unique>
  </element>
  <complexType name="RequirementType">
    <sequence>
      <element name="class" type="string"/>
      <element minOccurs="0" name="property" type="string"/>
      <element maxOccurs="unbounded" name="profile">
        <complexType>
          <sequence>
            <element ref="pl:Profile"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
  <element name="Profile">
    <complexType>
      <sequence>
        <element name="identifier">
          <complexType>
            <sequence>
              <element ref="pl:ProfileIdentifier"/>
            </sequence>
          </complexType>
        </element>
        <element maxOccurs="unbounded" minOccurs="0" name="metadata">
          <complexType>
            <sequence>
              <element ref="pl:KeyValuePair"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
    <unique name="keyOfKeyValuePairInMetadata">
      <selector xpath="pl:metadata"/>
      <field xpath="pl:KeyValuePair/pl:key"/>
    </unique>
  </element>
  <element name="ProfileIdentifier">
```

```
    <complexType>
      <sequence>
        <element name="name" type="string"/>
        <element minOccurs="0" name="versionIndicator" type="string"/>
      </sequence>
    </complexType>
  </element>
  <element name="KeyValuePair">
    <complexType>
      <sequence>
        <element name="key" type="string"/>
        <element name="value" type="string"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

B.3. descriptorTargets XSD

NOTE	Only the relevant fragment of the whole ShapeChangeConfiguration XML Schema is shown.

```
<element name="descriptorTargets">
 <complexType>
  <sequence>
   <element maxOccurs="unbounded" minOccurs="0" name="DescriptorTarget">
    <complexType>
     <attribute name="appliesTo" default="all">
      <simpleType>
       <restriction base="string">
        <enumeration value="ontology"/>
        <enumeration value="class"/>
        <enumeration value="conceptscheme"/>
        <enumeration value="property"/>
        <enumeration value="all"/>
       </restriction>
      </simpleType>
     </attribute>
     <attribute name="target" type="string" use="required"/>
     <attribute name="template" type="string" use="required"/>
     <attribute default="langString" name="format">
      <simpleType>
       <restriction base="string">
        <enumeration value="langString"/>
        <enumeration value="IRI"/>
        <enumeration value="string"/>
       </restriction>
      </simpleType>
     </attribute>
```

```
     <attribute default="ignore" name="noValueBehavior">
      <simpleType>
       <restriction base="string">
        <enumeration value="ignore"/>
        <enumeration value="populateOnce"/>
       </restriction>
      </simpleType>
     </attribute>
     <attribute default="" name="noValueText" type="string"/>
     <attribute default="connectInSingleTarget" name="multiValueBehavior">
      <simpleType>
       <restriction base="string">
        <enumeration value="connectInSingleTarget"/>
        <enumeration value="splitToMultipleTargets"/>
       </restriction>
      </simpleType>
     </attribute>
     <attribute default=" " name="multiValueConnectorToken" type="string"/>
    </complexType>
   </element>
  </sequence>
 </complexType>
</element>
```

B.4. rdfMapEntries XSD

NOTE	Only the relevant fragment of the whole ShapeChangeConfiguration XML Schema is shown.

```
<element name="rdfMapEntries">
 <complexType>
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="sc:RdfTypeMapEntry"/>
   <element ref="sc:RdfPropertyMapEntry"/>
   <element ref="sc:rdfMapEntries"/>
  </choice>
 </complexType>
</element>
<element name="RdfTypeMapEntry" type="sc:RdfTypeMapEntryType"/>
<complexType name="RdfTypeMapEntryType">
 <sequence/>
 <attribute name="type" type="string" use="required">
  <annotation>
   <documentation>Name of a UML type</documentation>
  </annotation>
 </attribute>
 <attribute name="schema" type="string">
  <annotation>
   <documentation>The name of the application schema package to which the UML type
```

```
belongs. Used to avoid ambiguity in case that multiple schemas are being
processed.</documentation>
  </annotation>
 </attribute>
 <attribute name="target" type="string" use="required">
  <annotation>
   <documentation>IRI of the RDFS/OWL class or datatype to which the UML type shall be
mapped
    (e.g. "ex1:A"). Note: the value is expected to be given as a QName, with the
namespace
    prefix matching the namespace abbreviation of a namespace declared in the
    configuration.</documentation>
  </annotation>
 </attribute>
 <attribute default="class" name="targetType">
  <annotation>
   <documentation>Type of the target (class or datatype) to which the UML type will be
    mapped.</documentation>
  </annotation>
  <simpleType>
   <restriction base="string">
    <pattern value="datatype"/>
    <pattern value="class"/>
   </restriction>
  </simpleType>
 </attribute>
 <attribute name="rule" type="string" use="optional" default="*">
  <annotation>
   <documentation>The encoding rule to which this mapping applies. May be “*” to
indicate that
    the mapping applies to all encoding rules.</documentation>
  </annotation>
 </attribute>
</complexType>
<element name="RdfPropertyMapEntry" type="sc:RdfPropertyMapEntryType"/>
<complexType name="RdfPropertyMapEntryType">
 <sequence/>
 <attribute name="property" type="string" use="required">
  <annotation>
   <documentation>Name of a UML property, optionally scoped to a class from the
application schema (example: FeatureX::propertyY).</documentation>
  </annotation>
 </attribute>
 <attribute name="schema" type="string">
  <annotation>
   <documentation>The name of the application schema package to which the UML property
belongs. Used to avoid ambiguity in case that multiple schemas are being
processed.</documentation>
  </annotation>
 </attribute>
 <attribute name="target" type="string" use="optional">
```

```
  <annotation>
   <documentation>IRI of the RDF/OWL property to which the UML property shall be
mapped (e.g. "ex1:propZ"). Can be omitted or empty if the property shall not be
encoded. Note: the value is expected to be given as a QName, with the namespace prefix
matching the namespace abbreviation of a namespace declared in the
configuration.</documentation>
  </annotation>
 </attribute>
 <attribute name="range" type="string">
  <annotation>
   <documentation>Range to use in class expressions involving the target (RDF/OWL)
property. Note: the value is expected to be given as a QName, with the namespace
prefix matching the namespace abbreviation of a namespace declared in the
configuration.</documentation>
  </annotation>
 </attribute>
 <attribute name="rule" type="string" use="optional" default="*">
  <annotation>
   <documentation>The encoding rule to which this mapping applies. May be “*” to
indicate that
    the mapping applies to all encoding rules.</documentation>
  </annotation>
 </attribute>
</complexType>
```

B.5. rdfConversionParameters XSD

RDF conversion parameters provide instructions for the conversion of application schema classes and properties to RDF/OWL.

NOTE	Only the relevant fragment of the whole ShapeChangeConfiguration XML Schema is shown.

```
<element name="rdfConversionParameters">
 <complexType>
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="sc:StereotypeConversionParameter"/>
   <element ref="sc:TypeConversionParameter"/>
   <element ref="sc:PropertyConversionParameter"/>
   <element ref="sc:rdfConversionParameters"/>
  </choice>
 </complexType>
</element>
<element name="StereotypeConversionParameter"
type="sc:StereotypeConversionParameterType"/>
<complexType name="StereotypeConversionParameterType">
 <sequence/>
 <attribute name="wellknown" use="required">
  <annotation>
```

```
    <documentation>Stereotype that is well-known to ShapeChange</documentation>
   </annotation>
   <simpleType>
    <restriction base="string">
     <enumeration value="DataType"/>
     <enumeration value="FeatureType"/>
     <enumeration value="Enumeration"/>
     <enumeration value="CodeList"/>
     <enumeration value="Union"/>
     <enumeration value="Type"/>
     <enumeration value="BasicType"/>
    </restriction>
   </simpleType>
  </attribute>
  <attribute name="subClassOf" type="string" use="required">
   <annotation>
    <documentation>IRIs of classes of which UML types with this stereotype shall be
subClassOf. Multiple IRIs are separated with spaces. Note: the values are expected to
be given as QNames, with the namespace prefixes matching the namespace abbreviations
of the namespaces declared in the configuration.</documentation>
   </annotation>
  </attribute>
  <attribute name="rule" type="string" use="optional" default="*">
   <annotation>
    <documentation>The encoding rule to which this parameter applies. May be “*” to
indicate that
     the parameter applies to all encoding rules.</documentation>
   </annotation>
  </attribute>
 </complexType>
 <element name="TypeConversionParameter" type="sc:TypeConversionParameterType"/>
 <complexType name="TypeConversionParameterType">
  <sequence/>
  <attribute name="type" type="string" use="required">
   <annotation>
    <documentation>Name of a UML type</documentation>
   </annotation>
  </attribute>
  <attribute name="schema" type="string">
   <annotation>
    <documentation>The name of the application schema package to which the UML type
belongs. Used to avoid ambiguity in case that multiple schemas are being
processed.</documentation>
   </annotation>
  </attribute>
  <attribute name="subClassOf" type="string" use="required">
   <annotation>
    <documentation>IRIs of classes of which the UML type shall be a subClassOf.
Multiple IRIs are separated with spaces. Note: the values are expected to be given as
QNames, with the namespace prefixes matching the namespace abbreviations of the
namespaces declared in the configuration.</documentation>
```

```
  </annotation>
 </attribute>
 <attribute name="rule" type="string" use="optional" default="*">
  <annotation>
   <documentation>The encoding rule to which this parameter applies. May be “*” to
indicate that
    the parameter applies to all encoding rules.</documentation>
  </annotation>
 </attribute>
</complexType>
<element name="PropertyConversionParameter" type=
"sc:PropertyConversionParameterType"/>
<complexType name="PropertyConversionParameterType">
 <sequence/>
 <attribute name="property" type="string" use="required">
  <annotation>
   <documentation>Name of a UML property, optionally scoped to a class from the
application schema (example: FeatureX::propertyY).</documentation>
  </annotation>
 </attribute>
 <attribute name="schema" type="string">
  <annotation>
   <documentation>The name of the application schema package to which the UML property
belongs. Used to avoid ambiguity in case that multiple schemas are being
processed.</documentation>
  </annotation>
 </attribute>
 <attribute default="false" name="global" type="boolean">
  <annotation>
   <documentation>Specifies if the UML property shall be encoded as a global
    property</documentation>
  </annotation>
 </attribute>
 <attribute name="subPropertyOf" type="string">
  <annotation>
   <documentation>IRIs of RDF/OWL properties of which the RDF/OWL implementation of
the UML property shall be a subPropertyOf. Multiple IRIs are separated with spaces.
Note: the values are expected to be given as QNames, with the namespace prefixes
matching the namespace abbreviations of the namespaces declared in the
configuration.</documentation>
  </annotation>
 </attribute>
 <attribute name="target" type="string">
  <annotation>
   <documentation>Name of the target UML property (scoped to a class from the
application schema - example: FeatureX::propertyY), whose RDF/OWL implementation will
be used to implement this property.</documentation>
  </annotation>
 </attribute>
 <attribute name="targetSchema" type="string">
  <annotation>
```

```
   <documentation>The name of the application schema package to which the target
property belongs. Used to avoid ambiguity in case that multiple schemas are being
processed.</documentation>
  </annotation>
 </attribute>
 <attribute name="rule" type="string" use="optional" default="*">
  <annotation>
   <documentation>The encoding rule to which this parameter applies. May be “*” to
indicate that
    the parameter applies to all encoding rules.</documentation>
  </annotation>
 </attribute>
</complexType>
```

B.6. constraintMappings XSD

NOTE	Only the relevant fragment of the whole ShapeChangeConfiguration XML Schema is shown.

```
<element name="constraintMappings">
 <complexType>
  <sequence>
   <element maxOccurs="unbounded" minOccurs="0" name="ConstraintMapping">
    <complexType>
     <attribute name="constraintType" use="required">
      <simpleType>
       <restriction base="string">
        <enumeration value="Text"/>
        <enumeration value="FOL"/>
        <enumeration value="OCL"/>
       </restriction>
      </simpleType>
     </attribute>
     <attribute default="iso19150-2:constraint" name="target" type="string"/>
     <attribute name="template" type="string" use="required"/>
     <attribute default="" name="noValue" type="string"/>
     <attribute default=" " name="multiValueConnectorToken" type="string"/>
     <attribute default="string" name="format">
      <simpleType>
       <restriction base="string">
        <enumeration value="langString"/>
        <enumeration value="string"/>
       </restriction>
      </simpleType>
     </attribute>
    </complexType>
   </element>
  </sequence>
 </complexType>
</element>
```

Bibliography

[1] W3C: Shapes Constraint Language (SHACL), W3C Working Draft 28 January 2016, http://www.w3.org/TR/2016/WD-shacl-20160128/

[2] OGC (2015): OGC® Testbed 11 Aviation - Guidance on Using Semantics of Business Vocabulary and Business Rules (SBVR) Engineering Report, https://portal.opengeospatial.org/files/?artifact_id=63794

[3] W3C: OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation 11 December 2012, https://www.w3.org/TR/owl-primer

[4] OGC (2011): OGC® OWS-8 Cross Community Interoperability (CCI) Semantic Mediation Engineering Report, https://portal.opengeospatial.org/files/?artifact_id=46342

[5] OGC (2013): OGC® OWS-9 System Security Interoperability (SSI) UML-to-GML-Application-Schema (UGAS) Conversion Engineering Report, https://portal.opengeospatial.org/files/?artifact_id=51784

[6] Longley D., Sporny M., Kellogg G., Lanthaler M. (2016): JSON-LD Framing 1.0 - An Application Programming Interface for the JSON-LD Syntax, http://json-ld.org/spec/latest/json-ld-framing/ (accessed on September 26, 2016 - status at the time: Draft Community Group Specification)

[7] W3C: JSON-LD 1.0, A JSON-based Serialization for Linked Data, W3C Recommendation 16 January 2014, https://www.w3.org/TR/json-ld/

[8] IETF: The GeoJSON Format, IETF RFC 7946, https://tools.ietf.org/html/rfc7946