OGC Testbed-14

Application Schema-based Ontology Development Engineering Report

Table of contents

Publication Date: 2019-02-04

Approval Date: 2018-12-13

Submission Date: 2018-10-30

Reference number of this document: OGC 18-032r2

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D022

Category: OGC Public Engineering Report

Editor: Johannes Echterhoff

Title: OGC Testbed-14: Application Schema-based Ontology Development Engineering Report

---

OGC Engineering Report

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.

Chapter 1. Summary

This report enhances the understanding of the relationships between application schemas based on the Unified Modeling Language (UML) and ontologies based on the Web Ontology Language (OWL). The work documented in this report provides and improves tools and principled techniques for the development of Resource Description Framework (RDF) based schemas from ISO 19109-conformant application schemas.

1.1. Requirements & Research Motivation

The following requirements pertaining to the process of deriving an OWL ontology from an application schema in UML have been addressed by the work documented in this Engineering Report (ER):

- Analyze the domain knowledge specified through Object Constraint Language (OCL) constraints in terms of logically-equivalent RDF/OWL expressions and axioms. So far, the conversion process defined by ISO 19150-2 and extended in OGC Testbed-12 (for details, see the OGC Testbed-12 ShapeChange Engineering Report [http://docs.opengeospatial.org/per/16-020.html]) can transform an OCL constraint only to a simple OWL annotation property, with a textual value containing the constraint description. This information is primarily useful for human consumption. When OCL constraints are transformed to OWL expressions and axioms, the domain knowledge that is originally encoded in OCL is made machine-processible, specifically to reasoners. The intent is to improve inferencing results as well as the detection of inconsistencies in ontologies and RDF data.
- Determine how to enrich an application schema to define property characteristics and relationships that can be expressed in OWL, but typically not in UML. Examples are sub-property relationships, and whether a property is symmetric. Through such *property enrichment*, the application schema modelling and implemented ShapeChange-based conversion process supports a set of OWL expressions and axioms that would otherwise need to be added to the generated ontologies as part of a manual post-processing step.
- Enhance ShapeChange so that new RDF and/or OWL (from now on: RDF/OWL) properties can be added to an OWL ontology that is derived from an application schema in UML, and to relate RDF/OWL properties that result from conversion of UML properties in the application schema to these new properties via rdfs:subPropertyOf relationships. The goal is to improve the ontological definition of properties that have very similar semantics. A more detailed description of the background and the motivation behind this requirement is given in section 7.1.

1.2. Recommendations for Future Work

This ER describes specific aspects and enhancements of encoding an ISO 19109-conformant UML application schema as an OWL ontology.

The ER builds upon, and extends, the analysis and implementation of such an encoding that was performed in Testbed-12, and is documented in the Testbed-12 ShapeChange ER [http://docs.opengeospatial.org/per/16-020.html].

This ER analyzes the conversion of OCL constraints to logically-equivalent RDF/OWL expressions and axioms. It also documents how to enrich an application schema to define property characteristics and relationships that can be expressed in OWL, such as sub-property relationships, and whether a property is symmetric. These topics are not covered by ISO 19150-2, but could inform a future revision of that standard.

In addition, this ER documents how new RDF/OWL properties can be added to an OWL ontology that is derived from an application schema in UML using ShapeChange. The Testbed-14 use case for such an addition to the ontology was to improve the ontological definition of UML properties that have very similar semantics.

This ER provides useful knowledge to any OGC member who has an application schema in UML, and needs to convert that schema to a corresponding OWL ontology.

The following sections document work items that should be addressed next.

1.2.1. Improve conversion of OCL constraints to Schematron, using XSLT2

Testbed-14 documented a number of recommendations for writing OCL constraints that shall be translated to OWL expressions. A major aspect of these recommendations is to significantly increase the use of quantifications (exists(…) and forAll(…)) in OCL expressions. These quantifications are readily supported by OWL.

However, extensive use of such quantifications can be an issue for the translation to Schematron based on XSLT1 - which is what the OCL to Schematron conversion of ShapeChange currently supports. The reason is that Schematron with the XSLT1 query binding does not directly support quantifications. More specifically: XPath 1.0, which is used by XSLT1 and thus also used to define Schematron assertions, does not support them. The conversion by ShapeChange represents quantifications through equivalent XPath expressions:

- *x → exists(t|b(t))* is represented by an XPath expression like: *boolean(τ(x)[τ(b(.))])*
- *x → forAll(t|b(t))* is represented by an XPath expression like: *count(τ(x))=count(τ(x)[τ(b(.))])*

Especially the representation of forAll(…) can quickly lead to highly complex and potentially inefficient XPath expressions. A chain of forAll(…) statements would result in a deeply nested tree of count expressions.

The situation could significantly be improved if XSLT2 was used as Schematron query binding, since XPath 2.0 directly supports quantified expressions [https://www.w3.org/TR/xpath20/#id-quantified-expressions]. Future work should therefore consider enhancing the OCL to Schematron conversion capability of ShapeChange, to perform the conversion to Schematron based on XSLT2 instead of XSLT1.

1.2.2. Deriving SHACL with ShapeChange

In OGC Testbed-14, the focus of the analysis for converting OCL constraints was on OWL as the target language. As documented in Conversion of OCL Constraints and the Annex Conversion of NAS OCL Constraints to OWL, some OCL language constructs (e.g. variables, addition, subtraction) and specific OCL constraints (that instances of a specific class are generally not allowed) cannot be represented in OWL. However, it may be possible to encode these constraints as Shapes Constraint Language (SHACL) constraints, so that they can be checked for an RDF dataset.

Future work should therefore analyze the conversion of OCL constraints to SHACL. If it turns out that SHACL can represent OCL constraints, enhance ShapeChange accordingly. Another aspect could be to enhance ShapeChange to derive specifications of ontology subsets using SHACL.

1.2.3. Implement conversion of OCL constraints to OWL expressions

OGC Testbed-14 analyzed the conversion of OCL constraints to OWL. However, due to resource limitations, the conversion was not implemented. Future work should include an implementation of the conversion.

1.3. Document contributor contact points

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

Contacts

Name	Organization
Deborah L. Nichols	The MITRE Corporation
Paul Birkel	Geosemantic Resources LLC
Johannes Echterhoff (editor)	interactive instruments GmbH

1.4. 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: ISO 19109:2015, Geographic information - Rules for application schema, https://www.iso.org/standard/59193.html
- ISO: ISO 19150-2:2015, Geographic information — Ontology — Part 2: Rules for developing ontologies in the Web Ontology Language (OWL), https://www.iso.org/standard/57466.html
- ISO: ISO/IEC 19507:2012, Information technology - Object Management Group Object Constraint Language (OCL) - apart from introductory material identical to the OMG specification Object Constraint Language v2.3.1, https://www.omg.org/spec/OCL/2.3.1/PDF
- W3C: SPARQL 1.1 Overview, W3C Recommendation 21 March 2013, https://www.w3.org/TR/sparql11-overview/
- W3C: OWL 2 Web Ontology Language, Direct Semantics (Second Edition), W3C Recommendation 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/
- W3C: OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/

Chapter 3. Terms and definitions

3.1. Abbreviated terms

ER	Engineering Report
IRI	Internationalized Resource Identifier
ISO	International Organization for Standardization
NAS	NSG Application Schema
NEO	NSG Enterprise Ontology
NSG	U.S. National System for Geospatial Intelligence
OCL	Object Constraint Language
OGC	Open Geospatial Consortium
OWL	Web Ontology Language
RDF	Resource Description Framework
SHACL	Shapes Constraint Language
SPARQL	SPARQL Protocol and RDF Query Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	Extensible Stylesheet Language Transformations

Chapter 4. Overview

Chapter 5 analyzes the domain knowledge specified through OCL constraints accompanying an application schema in UML in order to determine logically-equivalent RDF/OWL expressions and axioms that would accompany a corresponding OWL-based ontology.

Chapter 6 determines how to enrich an application schema to define property characteristics and relationships that can be expressed in OWL, but typically not in UML.

Chapter 7 determines how generalized RDF and/or OWL properties can be added to an OWL ontology that is derived from an application schema in UML, and then how to relate RDF/OWL properties that result from conversion of UML properties in the application schema to those new properties via rdfs:subPropertyOf relationships.

Annex A documents XML Schema definitions for ShapeChange extensions that were specified in OGC Testbed-14.

Annex B documents the analysis results of converting NAS OCL constraints to OWL.

Chapter 5. Conversion of OCL Constraints

An application schema defines the semantics of content and logical structure of geographic data that is relevant for a set of applications or a single application. An application schema is typically modelled using UML as the conceptual schema language, following the rules defined by ISO 19103 and ISO 19109. While UML can be used to model classes, their attributes, and relationships with other classes, specific rules that a UML element or a set of elements must fulfill can often not be defined with UML alone. For example, UML alone cannot represent the condition that attribute1 of ClassA must have the value CodelistX::code1 if attribute2 has a value greater than 5. OCL constraints are one way to define such rules in a machine-readable way. The condition from the example can be expressed by the following OCL constraint:

```
context ClassA inv: attribute2 > 5 implies attribute1 = CodelistX::code1
```

An application schema can contain multiple OCL constraints. The constraints extend the specification of the geographic data that is defined by the schema.

When encoding an application schema as an OWL ontology, OCL constraints can currently be converted (by ShapeChange) to annotation properties and other RDF/OWL properties with textual content (through so-called constraint mappings). For more details, see the OGC Testbed 12 ShapeChange ER [http://docs.opengeospatial.org/per/16-020.html#rdf_cr_constraints].

The current approach has the drawback that domain knowledge represented by OCL constraints is not converted into a form that is useful for reasoners.

For example: The OCL constraint mentioned before can be converted into a restriction on ClassA like the following:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectIntersectionOf (
      DataSomeValuesFrom( DPE(attribute2) DatatypeRestriction( DT(typeOf_attribute2)
xsd:minExclusive "5"^^xsd:integer ) )
      DataAllValuesFrom( DPE(attribute2) DatatypeRestriction( DT(typeOf_attribute2)
xsd:minExclusive "5"^^xsd:integer ) )
    )
  )
  ObjectIntersectionOf (
    ObjectSomeValuesFrom ( OPE(attribute1)
      ObjectOneOf ( IND(CodelistX::code1) )
    )
    ObjectAllValuesFrom ( OPE(attribute1)
      ObjectOneOf ( IND(CodelistX::code1) )
    )
  )
)
```

NOTE The OWL expression from the example is given in OWL Manchester Syntax [https://www.w3.org/TR/owl2-manchester-syntax/], which is used in ontology editors such as Protégé. This chapter uses OWL Manchester Syntax as well as OWL Functional-Style Syntax [http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/] to represent examples of OWL expressions.

With such knowledge, a reasoner would know that OWL individuals of type ClassA whose attribute2 values are greater than 5 must have (the individual that represents) CodelistX::code1 as values of attribute1.

This knowledge might be used for:

- consistency checking: if an OWL individual of type ClassA with attribute2 values greater than 5 were assigned a value for attribute1 that is not CodelistX::code1, then there is an inconsistency present, and
- inferencing: if an OWL individual of type ClassA has attribute2 values greater than 5, then it can be assumed that it has at least one attribute1 value that is CodelistX::code1.

This section documents the results of an analysis on how OCL constraints can be converted to OWL expressions and axioms in a way that reasoners can make use of the domain knowledge that is represented by the constraints. First, this ER discusses the translation of OCL language constructs in general. Then the ER focuses on the conversion of the various types of OCL constraints that occur in the NAS.

NOTE The difference between *translation* and *conversion* in the context of this chapter is as follows:

- *Translation* is about the mapping of OCL language constructs and whole expressions to OWL language constructs and expressions.
- *Conversion* includes the translation of an OCL expression but is also concerned with the full representation of the OCL constraint in an OWL ontology, for example how the resulting OWL expression is integrated in the ontology. The representation of the OCL constraint documentation - typically a human readable text - would also be part of the *conversion*.

5.1. Translation of OCL Language

Finding a correct translation of OCL language constructs (for details, see the OCL specification) to OWL expressions and axioms has been a research topic for the scientific community. An extensive analysis is provided by [1]. It shows that some OCL expressions can be translated to OWL. However, not all OCL expressions can be translated. Most notably, OWL does not support the concept of variables in expressions [2]. Thus, OCL expressions like the following cannot be translated:

- *inv: propA.propB→forAll(x|x.propC > x.propD)*
- *inv: self.propA > self.propB*

This analysis of OWL translations of OCL language constructs focuses on the OCL language constructs that are supported by the OCL parser of ShapeChange. Table 1 lists the language constructs for which a translation to OWL exists. Table 2 lists the language constructs for which such a translation is not possible.

Table 1. OCL language constructs for which a translation to OWL exists

Ref	OCL language construct	Explanation	OWL translation
1	Variable access self Example: *inv: self.x > 10*	The current object in the context of which the expression shall hold.	Only supported at start of the expression.
2	Iterator variable access	The variable is assigned a current value from the path that leads to the iterator expression.	Only simple case of single variable with singular use in iterator condition can be supported. For example: *inv: propA.propB→forAll(x|x.propC > 40)* or *inv: propA.propB→forAll(x|x.propC→oclIsKindOf(SomeClass))*
3	Integer or real constants Example: *123* or *3.1415*		"123"^^xsd:integer, "3.1415"^^xsd:double
4	Boolean constants Example: *true* or *false*		"true"^^xsd:boolean, "false"^^xsd:boolean
5	String constants Example: *‘xxxxx’*		"xxxxx"^^xsd:string

Ref	OCL language construct	Explanation	OWL translation
6	Enumeration constants Example: *Type::value*		The translation depends on the encoding of the enumeration: • Encoding under *rule-owl-cls-iso191502Enumeration* [https://shapechange.net/targets/ontology/uml-rdfowl-based-isois-19150-2/#rule-owl-cls-iso191502Enumeration]: The literal created for Type::value • Encoding under *rule-owl-cls-enumerationAsCodelist* [https://shapechange.net/targets/ontology/uml-rdfowl-based-isois-19150-2/#rule-owl-cls-enumerationAsCodelist]: see codelist constants (next row)
7	Codelist constants Example: *Type::value*		The individual that represents Type::value
8	If expression Example: *if x then y else z endif*	If x evaluates to true then the value of the expression is y, otherwise z.	• Functional Syntax: *ObjectUnionOf (ObjectIntersectionOf (x y) ObjectIntersectionOf(ObjectComplementOf(x) z))* • Manchester Syntax: *(x and y) or (not(x) and z)*
9	Simple property navigation Example: *x.prop*	Access the values of prop, which is a property of x.	Is translated to an OWL property expression (object or data property, depending on the conversion of prop). Can only be converted if x is a variable - either self or an iterator variable. **NOTE** As explained here, there is no universally applicable translation for a chain of property navigation steps. An example of such an OCL expression is: *inv: self.prop1.prop2→size()=2*.

Ref	OCL language construct	Explanation	OWL translation
10	Operation call oclIsKindOf() Examples: 1. *self.prop.oclIsKindOf(y)* 2. *self.oclIsKindOf(y)*	1. The value of property prop is checked for being of type y or one of its subtypes. 2. Object self is checked for being of type y, or one of its subtypes.	Case 1: property value type check: • Functional Syntax: *ObjectAllValuesFrom (OPE(x) CE(y))* • Manchester Syntax: *x only y* Case 2: object type check: • Functional Syntax: _SubClassOf(CE(x) CE(y)) • Manchester Syntax: *x SubClassOf: y*
11	Operation call oclIsTypeOf() Examples: 1. *self.prop.oclIsTypeOf(y)* 2. *self.oclIsTypeOf(y)*	1. The value of property prop is checked for being of type y, but NOT one of its subtypes. 2. Object self is checked for being of type y, but NOT one of its subtypes.	Same as oclIsKindOf(). **NOTE** This translation does not fully represent the semantics of oclIsTypeOf(), which checks for a specific type - excluding its subtypes. However, since semantic applications typically include subtypes when defining conditions for a supertype, translating oclIsTypeOf() the same way that oclIsKindOf() is translated seems reasonable. **NOTE** With SPARQL, it is possible to query individuals of a certain type and exclude all the individuals that are also subtypes of that type (for details, see here [https://stackoverflow.com/questions/44503942/sparql-1-1-get-all-the-instances-of-a-specific-classes-but-do-not-include-any-i]).
12	Operation call oclAsType() Example: *x.oclAsType(y)*	The values of property x are cast to type y. The result is 'undefined' if this is not possible.	• Functional Syntax: *ObjectAllValuesFrom (x (ObjectIntersectionOf (y* restOfExpression *)))* - Where restOfExpression is determined by the expression following the operation call. • Manchester Syntax: *x only (y and rest)*

Ref	OCL language construct	Explanation	OWL translation
13	Relational operator =, <>, <, >, <=, >= Examples: 1. *inv: self.x$_1$→exists(v$_1$|v$_1$ < y$_1$)* 2. *inv: self.x$_2$→forAll(v$_2$|v$_2$ < y$_2$)* 3. *inv: self.x$_3$ < y$_3$*		The examples show that three different approaches for quantifying the condition with relational operator exist: 1. The first operand is a variable defined in an existential quantification. 2. Same as before, but with a universal quantification. 3. Here, a quantification is not explicitly defined. The OCL expression assumes that x$_3$ has a value, and that it is smaller than y$_3$. If x$_3$ has multiple values, then all need to be smaller than y$_3$. In order to represent this in OWL, a logical combination - using an intersection class expression - of existential and universal quantification is needed. The translation of the relational operator would be as follows: • x is a data property, y is a literal: ◦ = ▪ Functional Syntax: *DataOneOf (y)* ▪ Manchester Syntax: *{ y }* ▪ Example for case 1 (existential quantification): *DataSomeValuesFrom (x DataOneOf (y))* ◦ <> ▪ Functional Syntax: *DataComplementOf (DataOneOf (y))* ▪ Manchester Syntax: *not { y }* ▪ Example for case 2 (universal quantification): *DataAllValuesFrom (x DataComplementOf (DataOneOf (y)))* ◦ < ▪ Functional Syntax: *DatatypeRestriction (datatypeOfX xsd:maxExclusive y)* ▪ Manchester Syntax: *datatypeOfX[< y]* ▪ Example for case 3 (existential and universal quantification): *ObjectIntersectionOf (*

Ref	OCL language construct	Explanation	OWL translation
14	Operation call size() Example: *x.size()*	Compute the number of characters in the string instance x.	If used in combination with relational operator and non-negative integer: • x.size() <= y ◦ Functional Syntax: *DataAllValuesFrom (x DatatypeRestriction (datatypeOfX xsd:maxLength y))* ◦ Manchester Syntax: *x only dataTypeOfX[xsd:maxLength y]* ◦ NOTE: *datatypeOfX* is a placeholder for the data type of property x. The word is not part of the OWL functional style or Manchester syntax. ◦ Same for '<', with y decreased by 1. • x.size() >= y ◦ Functional Syntax: *DataAllValuesFrom (x DatatypeRestriction (datatypeOfX xsd:minLength y))* ◦ Manchester Syntax: *x only dataTypeOfX[xsd:minLength y]* ◦ Same for '>', with y increased by 1. • x.size() = y ◦ Functional Syntax: *DataAllValuesFrom (x DatatypeRestriction (datatypeOfX xsd:length y))* ◦ Manchester Syntax: *x only dataTypeOfX[xsd:length y]*

Ref	OCL language construct	Explanation	OWL translation
15	Operation call and, or, xor, implies	Logical combination of expressions	• x and y: ◦ Functional Syntax: *ObjectIntersectionOf(x y)* ◦ Manchester Syntax: *(x) and (y)* • x or y: ◦ Functional Syntax: *ObjectUnionOf(x y)* ◦ Manchester Syntax: *(x) or (y)* • x implies y: ◦ Functional Syntax: *ObjectUnionOf(ObjectComplementOf(x) y)* ◦ Manchester Syntax: *not(x) or (y)* • x xor y: xor is not directly supported by OWL. However, *xor* can be expressed through a combination of logical *and*, logical *or*, and *negation* (e.g. through *(not(x) and y) or (x and not(y))*).

Ref	OCL language construct	Explanation	OWL translation
16	Set operation call size() Example: *x→size()*	Number of objects in the set of values on the UML property x. NOTE: Since the OCL collect() operation is not supported by OWL (for further details, see here), the OCL set operation call size() can only be translated when checking the number of values of a particular property. Therefore, x has to represent a UML property.	If used in combination with relational operator and non-negative integer: • x→size() <= y: ◦ Functional Syntax: *ObjectMaxCardinality (y x)* ◦ Manchester Syntax: *x max y* ◦ Same for '<', with y decreased by 1. • x→size() >= y: ◦ Functional Syntax: *ObjectMinCardinality (y x)* ◦ Manchester Syntax: *x min y* ◦ Same for '>', with y increased by 1. • x→size() = y: ◦ Functional Syntax: *ObjectExactCardinality (y x)* ◦ Manchester Syntax: *x exactly y* **NOTE** If x is represented by a DataProperty, use DataMaxCardinality, DataMinCardinality and DataExactCardinality instead.
17	Set operation call isEmpty() Example: *x→isEmpty()*	Predicate: Is the set represented by x empty?	If x identifies a property: translate with equivalent expression *x→size()=0* If x is a variable (e.g. 'self'): ignore. If self→isEmpty() then the class would be equivalent to owl:Nothing. However, then the class would not be satisfiable, which will likely cause consistency errors reported by a reasoner.

Ref	OCL language construct	Explanation	OWL translation
18	Set operation call notEmpty() Example: *x→notEmpty()*	Predicate: Is the set represented by prop not empty?	If x identifies a property, then translate with equivalent expression *x→size()>=1*. It would also be possible to translate this with an existential quantification: *ObjectSomeValuesFrom (OPE(x typeOfX))* (in case that x is an object property) and *DataSomeValuesFrom (DPE(x datatypeOfX))* (in case that x is a data property). If x is a variable (e.g. 'self'): ignore the whole constraint. Checking that a given individual or literal is not empty makes no sense.
19	Iterator call exists() Example: *x→exists(t|b(t))*	Predicate: Does the set x contain an object t for which the Boolean expression b(t) holds?	Existential quantification *Object/DataSomeValuesFrom(OPE CE)*, where OPE is the translation of *t* (i.e., x) and CE is the translation of *b(t)*.
20	Iterator call forAll() Example: *x→forAll(t|b(t))*	Predicate: Does the set x only contain objects t for which the Boolean expression b(t) holds?	Universal quantification *Object/DataAllValuesFrom(OPE CE)*, where OPE is the translation of *t* (i.e., x) and CE is the translation of *b(t)*.
21	Pattern matching function on Strings Example: *x.matches(pattern)* Note: This operation call is an extension. It is not part of the OCL standard.	Boolean function which yields true if the pattern of type String matches the String argument.	x is a data property: • Functional Syntax: *DataAllValuesFrom (x DatatypeRestriction (datatypeOfX xsd:pattern y)* • Manchester Syntax: *x only datatypeOfX[xsd:pattern y]*

Ref	OCL language construct	Explanation	OWL translation
22	Iterator call select() Example: *x→select(t|b(t))*	Compute the set of those objects t in x, for which the predicate b(t) holds.	select() can be translated for particular cases, through equivalent OCL expresions that use existential or universal quantification: • *x→select(t|b(t))→notEmpty()* is equivalent to *x→exists(t|b(t))* • *x→select(t|b(t))→isEmpty()* is equivalent to *x→forAll(t|not b(t))* • *x→select(t|b_1(t))→exists(t|b_2(t))* is equivalent to *x→exists(t|b_1(t) and b_2(t))* • *x→select(t|b_1(t))→forAll(t|b_2(t))* is equivalent to *x→forAll(t|b_1(t) implies b_2(t))* The last two cases work, because the type of t in the expression with select() is the same for the select() and the following quantification.

Table 2. OCL language constructs for which a translation to OWL is not possible

Ref	OCL language construct	Explanation	Reason why a translation to OWL is not possible
1	Let variable access and let expression Example: *let x=y in z(x)*	Assignment of expression y to variable x. Result is z(x).	A general translation of let variables and expressions in OWL is not possible because OWL does not support variables. For simple OCL expressions, replacing the expressions bound to the let variables wherever the variables are used in the OCL expression could lead to an OCL expression that can be translated to OWL. **NOTE** NAS OCL constraints currently do not make use of let variables and expressions.

Ref	OCL language construct	Explanation	Reason why a translation to OWL is not possible
2	Property navigation using collect or shorthand for collect Example: *self→collect(prop1)→collect(prop2)→collect(prop3)*, which is the same as *self.prop1.prop2.prop3* (which uses the shorthand notation for collect)	The OCL specification defines the "Collect" operation. The purpose of that operation is to derive a collection from some other collection. The OCL specification also defines a shorthand notation for the collect operation, to simplify navigation through multiple objects in an OCL expression. In the example, prop1 represents the collection of values of property prop1 that belongs to the current object (self). From the values of prop1, we then collect the values of prop2, and from there we collect the values of prop3. Note: If prop1 and/or prop2 represent a collection, then the result is a bag of values.	OWL does not have an equivalent for "collecting" the values of properties on arbitrary levels of an object structure. OWL supports a number of class expressions, such as property restrictions, cardinality restrictions, and logical combinations of expressions. Property restrictions such as existential and universal quantifications can sometimes be used for creating an OWL expression that defines the same intent as an OCL constraint that uses the collect operation. For example: • The OCL constraint *inv: self.prop1.prop2→notEmpty()* can be represented by the OWL expression *ObjectSomeValuesFrom (OPE(prop1) ObjectSomeValuesFrom (OPE(prop2) CE(typeOf_prop2)))*. • The OCL constraint *inv: self.prop1.prop2→isEmpty()* is the same as *inv: self.prop1.prop2→size()=0*, which can be represented by the OWL expression *ObjectAllValuesFrom (OPE(prop1) ObjectExactCardinality (0 OPE(prop2)))*. • The OCL constraint *inv: self.prop1.prop2→forAll(x|x = SomeCodeList::SomeCode)* can be represented by the OWL expression *ObjectAllValuesFrom (OPE(prop1) ObjectAllValuesFrom (OPE(prop2) ObjectOneOf (IND(SomeCodeList::SomeCode))))* The following OCL constraint cannot be represented: • *inv: self.prop1.prop2→size()=2* cannot be represented because an OWL cardinality restriction is always evaluated for a specific property of a given class. With the OWL expression *ObjectExactCardinality (2 OPE(prop2))*, one can check that for a given object (here: a value of prop1), prop2 has two

Ref	OCL language construct	Explanation	Reason why a translation to OWL is not possible
3	Operation call allInstances() Example: *ClassX.allInstances()*	Set of all object instances of type ClassX.	OWL does not support accessing the collection of all objects of a given type. For example, it is not possible to express a constraint like *inv: FeatureTypeX.allInstances()→size()>1000*.
4	Operation call +,-,*,/	Value of x+y, etc.	OWL does not support arithmetic operations.
5	Operation call concat() Example: *x.concat(y)*	String concatenation of x and y.	OWL does not support the concatenation of string literals.
6	Operation call substring() Example: *x.substring(y,z)*	Substring of x running from position y to position z	OWL does not support subsetting of string literals.
7	Iterator call isUnique() Example: *x→isUnique(t|y(t))*	Predicate: Does the set x only contain objects t for which the expression y(t) creates mutually different objects?	Not applicable in OWL. OWL is based on set-theory. As such, it has a set-based view on property values (which can be individuals or literals). OWL does not have concepts to represent specific types of collections of values (like list, bag, and sequence). It also does not have concepts to check if the values of a property are unique. If the RDF encoded resource had duplicate values for a property, those would be ignored by OWL applications. An OWL editor like Protégé typically prevents creation of an assertion that connects individual A via property P to individual B if the same combination (A,P,B) has already been asserted.

5.2. Attaching the resulting OWL expression to a class

The OWL expression corresponding to an OCL constraint is defined as an OWL restriction on the OWL class that represents the UML class which provides the context for the OCL constraint.

The following two listings show the full OWL restriction resulting from the initial OCL constraint example - first in Turtle encoding, then in RDF/XML encoding:

Listing 1. OWL restriction resulting from initial OCL constraint example - Turtle encoding

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://example.org/ClassA>
  a owl:Class ;
  rdfs:subClassOf [
    a owl:Class ;
    owl:unionOf (
     _:genid2
     _:genid12
   )
  ] .

_:genid2
  a owl:Class ;
  owl:intersectionOf (
   _:genid4
   _:genid8
 ) .

_:genid4
  a owl:Restriction ;
  owl:onProperty <http://www.example.org/attribute1> ;
  owl:someValuesFrom [
    a owl:Class ;
    owl:oneOf ( <http://www.example.org/CodelistX/code1> )
  ] .

_:genid8
  a owl:Restriction ;
  owl:onProperty <http://www.example.org/attribute1> ;
  owl:allValuesFrom [
    a owl:Class ;
    owl:oneOf ( <http://www.example.org/CodelistX/code1> )
  ] .

_:genid12
  a owl:Class ;
  owl:complementOf [
    a owl:Class ;
    owl:intersectionOf (
     _:genid15
     _:genid20
   )
  ] .

_:genid15
  a owl:Restriction ;
  owl:onProperty <http://www.example.org/attribute2> ;
  owl:someValuesFrom [
    a rdfs:Datatype ;
    owl:onDatatype xsd:double ;
    owl:withRestrictions ( _:genid18 )
  ] .

_:genid18 xsd:minExclusive 5.000000e+0 .
_:genid20
  a owl:Restriction ;
  owl:onProperty <http://www.example.org/attribute2> ;
  owl:allValuesFrom [
    a rdfs:Datatype ;
    owl:onDatatype xsd:double ;
    owl:withRestrictions ( _:genid23 )
  ] .

_:genid23 xsd:minExclusive 5.000000e+0 .
```

Listing 2. OWL restriction resulting from initial OCL constraint example - RDF/XML encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<owl:Class rdf:about="http://example.org/ClassA"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:owl=
"http://www.w3.org/2002/07/owl#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#" xmlns:rdfs=
"http://www.w3.org/2000/01/rdf-schema#">
 <rdfs:subClassOf>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
     <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
       <owl:onProperty rdf:resource="http://www.example.org/attribute1"/>
       <owl:someValuesFrom>
        <owl:Class>
         <owl:oneOf rdf:parseType="Collection">
          <rdf:Description rdf:about="http://www.example.org/CodelistX/code1"/>
         </owl:oneOf>
        </owl:Class>
       </owl:someValuesFrom>
      </owl:Restriction>
      <owl:Restriction>
       <owl:onProperty rdf:resource="http://www.example.org/attribute1"/>
       <owl:allValuesFrom>
        <owl:Class>
         <owl:oneOf rdf:parseType="Collection">
          <rdf:Description rdf:about="http://www.example.org/CodelistX/code1"/>
         </owl:oneOf>
        </owl:Class>
       </owl:allValuesFrom>
```

```
        </owl:Restriction>
       </owl:intersectionOf>
      </owl:Class>
      <owl:Class>
       <owl:complementOf>
        <owl:Class>
         <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
           <owl:onProperty rdf:resource="http://www.example.org/attribute2"/>
           <owl:someValuesFrom>
            <rdfs:Datatype>
             <owl:onDatatype rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
             <owl:withRestrictions rdf:parseType="Collection">
              <rdf:Description>
               <xsd:minExclusive rdf:datatype="http://www.w3.org/2001/XMLSchema#double"
>5.0</xsd:minExclusive>
              </rdf:Description>
             </owl:withRestrictions>
            </rdfs:Datatype>
           </owl:someValuesFrom>
          </owl:Restriction>
          <owl:Restriction>
           <owl:onProperty rdf:resource="http://www.example.org/attribute2"/>
           <owl:allValuesFrom>
            <rdfs:Datatype>
             <owl:onDatatype rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
             <owl:withRestrictions rdf:parseType="Collection">
              <rdf:Description>
               <xsd:minExclusive rdf:datatype="http://www.w3.org/2001/XMLSchema#double"
>5.0</xsd:minExclusive>
              </rdf:Description>
             </owl:withRestrictions>
            </rdfs:Datatype>
           </owl:allValuesFrom>
          </owl:Restriction>
         </owl:intersectionOf>
        </owl:Class>
       </owl:complementOf>
      </owl:Class>
     </owl:unionOf>
    </owl:Class>
   </rdfs:subClassOf>
  </owl:Class>
```

5.3. Recommendations for writing OCL constraints for subsequent translation to OWL expressions

When writing OCL constraints that shall be converted to OWL expressions, take the following recommendations into account:

- Do not use a sequence of collect operations like *self→collect(prop1)→collect(prop2)* - or the shorthand notation (*self.prop1.prop2*). Instead, use single navigation steps and quantifications that explicitly define the intent, for example: *self.prop1→exists(p1|p1.prop2→forAll(p2|p2 > 3)*
- Be aware that a constraint like *self.prop3 > 10*, with prop3 having multiplicity 0..*, means that prop3 must have at least one value and all values must be greater than 10.
  - If the intent is to allow prop3 having no value, then use one of the following expressions:
    - *self.prop3→forAll(p|p > 10)*
    - *self.prop3→notEmpty() implies self.prop3 > 10*
  - If the intent is that prop3 shall have at least one value that is greater than 10, use: *self.prop3→exists(p|p > 10)*
  - If prop3 may either have no value, or a set of values for which at least one shall be greater than 10, use:
    - *self.prop3→notEmpty() implies self.prop3→exists(p|p > 10)* or (the equivalent OCL expression)
    - *self.prop3→isEmpty() or self.prop3→exists(p|p > 10)*

NOTE Adhering to these recommendations would facilitate a translation of OCL constraints to OWL class expressions. However, translating such OCL constraints to Schematron - to be used for validating XML data - based on XSLT1 (which is the approach that ShapeChange currently implements) can lead to odd and likely inefficient Schematron assertions. A Schematron implementation based on XSLT2 would be much more suited to handle these OCL constraints. This is documented in more detail in a future work item.

5.4. Considerations for translation optimization

When translating OCL to OWL, an analysis of an expression may lead to improved results. Consider the following two examples of OCL constraints:

- inv: self.prop→forAll(x|x = CodelistType::codeA or x = CodelistType::codeB or x = Codelisttype::codeC)
  - A direct translation of this expression would be:
    - Functional Syntax: *ObjectAllValuesFrom(OPE(prop) ObjectUnionOf(ObjectOneOf(IND(Codelisttype::codeA)) ObjectOneOf(IND(Codelisttype::codeB)) ObjectOneOf(IND(Codelisttype::codeC)))*
    - Manchester Syntax: *prop only ({codeA} or {codeB} or {codeC})*
  - However, if the translator was able to identify that the forAll() actually defines a restriction of the value of 'prop' to a set of codes from a single code list, then the OCL constraint can also be expressed in OWL as:
    - Functional Syntax: *ObjectAllValuesFrom(OPE(prop) ObjectOneOf(IND(Codelisttype::codeA) IND(Codelisttype::codeB) IND(Codelisttype::codeC)))*
    - Manchester Syntax: *prop only ({codeA, codeB, codeC})*

- inv:self.prop >= 0 and self.prop < 360
  - The constraint restricts the range for prop to [0,360).
  - The direct translation of this expression would be:
    - Functional Syntax: *ObjectIntersectionOf(ObjectIntersectionOf(DataSomeValuesFrom(DPE(prop) DatatypeRestriction(DT(typeOf_prop) xsd:minInclusive "0.0"^^xsd:double)) DataAllValuesFrom(DPE(prop) DatatypeRestriction(DT(typeOf_prop) xsd:minInclusive "0.0"^^xsd:double))) ObjectIntersectionOf(DataSomeValuesFrom(DPE(prop) DatatypeRestriction(DT(typeOf_prop) xsd:maxExclusive "360.0"^^xsd:double)) DataAllValuesFrom(DPE(prop) DatatypeRestriction(DT(typeOf_prop) xsd:maxExclusive "360.0"^^xsd:double))))*
    - Manchester Syntax: *((prop some xsd:double[>=0]) and (prop only xsd:double[>=0])) and ((prop some xsd:double[<360]) and (prop only xsd:double[<360]))*
  - If the translator was able to identify that the constraint defines a range restriction, this could also be expressed as:
    - Functional Syntax: *ObjectIntersectionOf(DataSomeValuesFrom(OPE(prop) DatatypeRestriction(DT(typeOf_prop) xsd:minInclusive "0.0"^^xsd:double xsd:maxExclusive "360.0"^^xsd:double)) DataAllValuesFrom(OPE(prop) DatatypeRestriction(DT(typeOf_prop) xsd:minInclusive "0.0"^^xsd:double xsd:maxExclusive "360.0"^^xsd:double)))*
    - Manchester Syntax: *(prop some xsd:double[>=0, < 360]) and (prop only xsd:double[>=0, < 360])*

5.5. Conversion of NAS OCL Constraints using ShapeChange

The NAS uses a variety of OCL constraints to specify restrictions for types defined in the application schema. Most of the different types of NAS OCL constraints can be translated to OWL. However, some of the constraints cannot be translated due to the restrictions of OWL documented in section Translation of OCL Language, specifically: OWL does not have a concept for defining variables and it does not support arithmetic operations (like subtraction).

Furthermore, some NAS OCL constraints serve as placeholders for future modelling work. One of them addresses a limitation of ShapeChange: changes to the model structure, like transformation of association classes and flattening of complex types, typically do not change the structure of OCL constraints. Thus, such changes can lead to OCL constraints becoming invalid. ShapeChange can detect OCL constraints that have become invalid and inform the user of that situation (for further details, see the documentation of ShapeChange [https://shapechange.net/transformations/common-transformer-functionality/#Parsing_and_Validating_Constraints]). Invalid OCL constraints are typically converted to text constraints. In any case, an automated conversion of invalid OCL constraints to OWL would not lead to useful results. Thus, the implications of model transformations for a subsequent conversion to OWL, including a conversion of OCL constraints, need to be taken into account.

NOTE

In OGC Testbed-12, the ShapeChange process for creating the NEO contained two transformations, both of which changed the model structure:

- Mapping association classes [http://docs.opengeospatial.org/per/16-020.html#rdf_cr_associationclass] and
- Replacing property with union as value type with union options [http://docs.opengeospatial.org/per/16-020.html#rdf_cr_class_union] (see second alternative).

The detailed results of an analysis regarding the conversion of categories of current NAS OCL constraints to OWL are documented in Annex B.

Chapter 6. OWL Property Enrichment

When converting a UML property to an OWL property, ShapeChange determines - based upon the configuration (encoding rule, parameters, etc.):

- the name and scope (local vs. global) [http://docs.opengeospatial.org/per/16-020.html#rdf_cr_properties_scope] of the OWL property;
- its domain and range; and
- if, as well as how, descriptive information available for the property is represented (e.g. as skos:definition and rdfs:label).

OWL 2 includes axioms that can be used to further characterize an OWL property and to establish specific logical relationships with other OWL properties. Table 3 documents these types of axioms. The abbreviation 'OPE' represents an OWL object property expression, while the abbreviation 'DPE' represents an OWL data property expression. The abbreviation 'PE' is used where it is possible to describe the property axiom in terms applicable to both kinds of OWL property.

Table 3 also indicates the applicability of the OWL 2 property axioms for the three distinct cases in which OWL properties are used to represent information from a UML model. An OWL Object Property may represent a UML association role that relates two individual entities. An OWL Object Property may instead represent a UML attribute that has a value which is a complex datatype; this is necessary because complex datatypes are represented in OWL by OWL classes (see ISO 19150-2, Section 6.6.3.2). An OWL Data Property is used to represent a UML attribute whose value is a simple datatype. The application of OWL 2 property axioms to property expressions therefore differs due to the difference in the ranges of the property in the three cases.

The first four OWL property axioms listed in Table 3 relate two or more properties, while the remaining axioms apply to a single property.

The OWL 2 property axioms do not include an inverse functional axiom for data properties, even though certain data properties (e.g., those whose values are unique identifiers) appear to meet the criterion for inverse-functionality.

Table 3. Overview of OWL property axioms relevant for property enrichment

Name	Description	Practical Importance	Applicable for OWL Object Property representing a UML association role	Applicable for OWL Object Property representing a UML attribute having a complex datatype	Applicable for OWL Data Property representing a UML attribute having a simple datatype
SubPropertyOf	All the superproperties of a property PE, such that the extension of PE is included in the extension of the superproperty. If OPE_1 is a subproperty of OPE_2, then any individual *x* that is connected by OPE_1 to an individual *y* is also connected to *y* by OPE_2. If DPE_1 is a subproperty of DPE_2, then any individual *x* that is connected by DPE_1 to a literal *y* is also connected to *y* by DPE_2.	Inference (infer a more general relationship from a specialized one)	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Object_Subproperties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Object_Subproperties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Data_Subproperties]
Equivalent	All the property expressions PE_i, where $1 \le i \le n$, are semantically equivalent to each other. A substitution of equivalent properties preserves meaning. The extension of each property contains the same tuples.	Inference (mapping)	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Equivalent_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Equivalent_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Equivalent_Data_Properties]
Disjoint	All the property expressions PE_i, where $1 \le i \le n$, are pairwise disjoint. For disjoint object properties OPE_i and OPE_j, no individual *x* can be connected to an individual *y* by both OPE_i and OPE_j (for $i \neq j$). For disjoint data properties, DPE_i and DPE_j, no individual *x* can be connected to a literal *l* by both DPE_i and DPE_j, (for $i \neq j$). The extensions of disjoint properties do not overlap.	Inference (constraint)	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Disjoint_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Disjoint_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Disjoint_Data_Properties]

Name	Description	Practical Importance	Applicable for OWL Object Property representing a UML association role	Applicable for OWL Object Property representing a UML attribute having a complex datatype	Applicable for OWL Data Property representing a UML attribute having a simple datatype
Inverse	All the properties OPE_i, where $2 \le i \le n$, such that if any individual *x* is connected by OPE_1 to an individual *y*, then also *y* is connected by OPE_i to *x*, and vice versa. If multiple inverses exist for OPE_1, then all the OPE_i are equivalent properties.	Inference (infer implicit "reverse" relationship from an asserted relationship)	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Inverse_Object_Properties]	no	no
Functional	If a property is functional, then an individual *x* may be related to at most one distinct individual *x* (for a functional object property OPE) or at most one distinct literal *l* (for a functional data property DPE). A functional property PE has a maximum cardinality of 1.	Integrity constraint	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Functional_Object_Properties]	no	no
Inverse-Functional	An object property expression OPE is inverse-functional if and only if, for each individual *x*, there can be at most one individual *y* such that *y* is connected by OPE with *x*.	Integrity constraint	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Inverse-Functional_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Inverse-Functional_Object_Properties]	no (unsupported)

Name	**Description**	**Practical Importance**	**Applicable for OWL Object Property representing a UML association role**	**Applicable for OWL Object Property representing a UML attribute having a complex datatype**	**Applicable for OWL Data Property representing a UML attribute having a simple datatype**
Reflexive	An object property OPE is reflexive if and only if each individual *x* is connected by OPE to itself.	Inference	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Reflexive_Object_Properties]	no	no
Irreflexive	An object property OPE is irreflexive if and only if no individual *x* is connected by OPE to itself.	Inference	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Irreflexive_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Irreflexive_Object_Properties]	no
Symmetric	An object property OPE is symmetric if and only if, for each individual *x* that is connected by OPE to an individual *y*, then also *y* is connected by OPE to *x*.	Inference (constraint)	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Symmetric_Object_Properties]	no	no

Name	Description	Practical Importance	Applicable for OWL Object Property representing a UML association role	Applicable for OWL Object Property representing a UML attribute having a complex datatype	Applicable for OWL Data Property representing a UML attribute having a simple datatype
Asymmetric	An object property OPE is asymmetric if and only if, for each individual *x* that is connected by OPE to *y*, then *y* is not connected by OPE to *x*.	Inference (constraint)	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Asymmetric_Object_Properties]	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Asymmetric_Object_Properties]	no
Transitive	An object property OPE is transitive if and only if, if an individual *x* is connected by OPE to an individual *y*, and *y* is connected by OPE to an individual *z*, then *x* is also connected by OPE to *z*.	Inference	yes [https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Transitive_Object_Properties]	no	no

NOTE

Antisymmetry is another characteristic of a property:

If individuals x and y are related by antisymmetric property P, and also y and x are related by P, then x is identical to y.

Or, from Wikipedia [https://en.wikipedia.org/wiki/Antisymmetric_relation]:

> R is anti-symmetric precisely if for all a and b in X:
>
> - if R(a,b) and R(b,a), then a = b, or, equivalently,
> - if R(a,b) with a ≠ b, then R(b,a) must not hold.

Antisymmetry is different from asymmetry. A property is asymmetric if it is both antisymmetric and irreflexive.

Antisymmetry is important in mathematics and set theory, for example.

OWL does not define a property axiom for antisymmetry. From a discussion on the W3C OWL developer mailing list (see the thread with subject "Semantics of antisymmetric properties", starting with this message [http://lists.w3.org/Archives/Public/public-owl-dev/2007JanMar/0195.html]), the reason appears to be that it is (or at least was) unknown if reasoning with antisymmetric properties is decidable.

In order to enrich properties in the UML model with indicators for these OWL property axioms, tagged values (identified by italicization) are used:

- If an axiom represents a relationship with other properties, then the value of the tag is a comma-separated list of property identifiers. Such an identifier either identifies a UML property from the application schema, or an RDF/OWL property.
  - A UML property is identified by its full package-qualified name, starting with the application schema package. For example: "Some Application Schema::Some Subpackage::Another Subpackage::SomeClass::aProperty".
    - NOTE: "::" is used as separator between the names of packages, a class, and a property.
    - NOTE: If a tagged value identifies a UML property by its full package-qualified name from the original model, and a ShapeChange transformation that is part of the workflow to derive an ontology modifies that name, then when producing the ontology ShapeChange will not be able to find the UML property that is identified by the tagged value. In such a situation, ShapeChange will log a warning to inform the user that the property axiom could not be created as intended.
  - A QName-like string is used to identify an RDF/OWL property. For example: "ex:Property". The namespace abbreviation must belong to one of the namespaces in the OWL target configuration of ShapeChange.
- If an axiom defines a characteristic of an OWL property, then the name of the axiom is part of a comma-separated list of names in tagged value *owlLogicalCharacteristics*.
  - NOTE: ShapeChange will parse the names defined by the tagged value to identify the names of the property axioms that apply to the UML property. When parsing a name, both the case

of characters and non-word characters (like the dash in 'Inverse-Functional') will be ignored.

The following table documents which tagged value is used to model a given property axiom, and the type of the property axiom (either defining a characteristic, or a property relationship).

Table 4. Modeling OWL Property Axioms

Name of OWL Property Axiom	**Type (characteristic, or property relationship)**	**Name of tagged value**
SubPropertyOf	property relationship (0..*)	*owlSubPropertyOf*
Equivalent	property relationship (0..*)	*owlEquivalentProperties*
Disjoint	property relationship (0..*)	*owlDisjointProperties*
Inverse	property relationship (0..*)	*owlInverseProperties*
Functional	characteristic	*owlLogicalCharacteristics*
Inverse-Functional	characteristic	*owlLogicalCharacteristics*
Reflexive	characteristic	*owlLogicalCharacteristics*
Irreflexive	characteristic	*owlLogicalCharacteristics*
Symmetric	characteristic	*owlLogicalCharacteristics*
Asymmetric	characteristic	*owlLogicalCharacteristics*
Transitive	characteristic	*owlLogicalCharacteristics*

NOTE

- ShapeChange supports mapping of tag names while loading a model. For further details, see the documentation of tag aliases [http://shapechange.net/get-started/config/input/#Tagaliases]. This can be used to map a domain-specific tag name to a tag name that ShapeChange understands (e.g. *owlLogicalCharacteristics*).
- SubPropertyOf relationships can also be defined through the ShapeChange configuration, via *PropertyConversionParameter*.
- The following axioms are mutually exclusive: a) functional and inverse-functional, b) reflexive and irreflexive, c) symmetric and asymmetric. If tag *owlLogicalCharacteristics* contains mutually exclusive axioms, ShapeChange will issue a warning - and encode both axioms (since there is no way to tell what is correct in such a situation)!

The ShapeChange behavior to perform property enrichment is controlled via *rule-owl-prop-propertyEnrichment*.

Chapter 7. OWL Property Generalization

7.1. Background and Motivation

The conversion of UML properties defined in an application schema, to RDF/OWL properties specified in an ontology, is described in the OGC Testbed 12 ShapeChange ER [http://docs.opengeospatial.org/per/16-020.html#rdf_cr_properties]. ShapeChange provides various configuration options to control the conversion of UML properties, for example to derive an RDF/OWL property with global or local scope, and to define of which RDF/OWL properties the resulting RDF/OWL property is a sub-property. However, the current conversion functionality does not include the creation of new OWL object and data properties.

The NSG Application Schema (NAS) includes feature types (UML classes) with properties of the same name, for example "featureFunction". See the following two figures, where the property appears in feature type Building and feature type Cableway (other feature properties have been omitted for brevity).

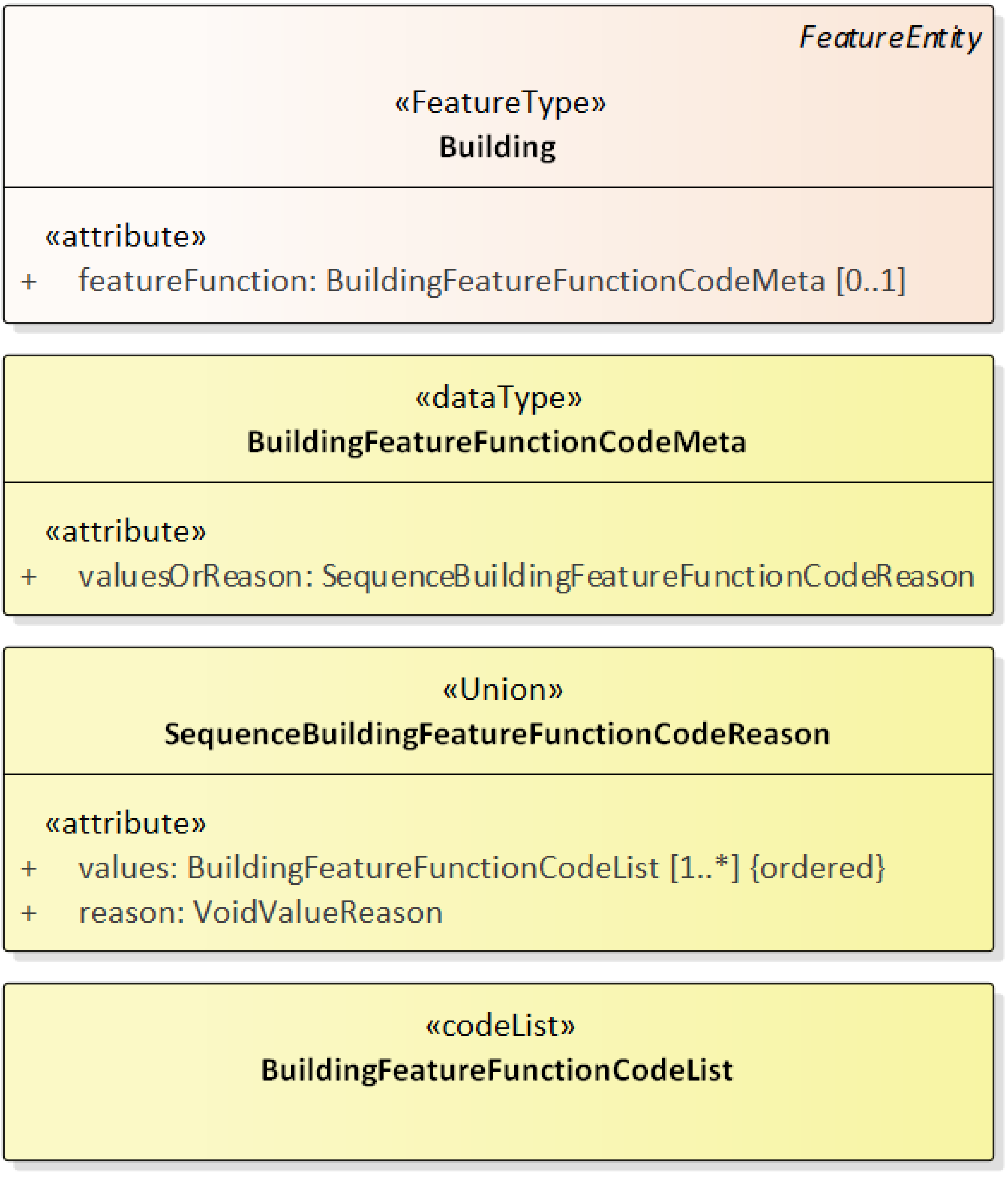
FeatureEntity
«FeatureType»
Building
«attribute»
+ featureFunction: BuildingFeatureFunctionCodeMeta [0..1]
«dataType»
BuildingFeatureFunctionCodeMeta
«attribute»
+ valuesOrReason: SequenceBuildingFeatureFunctionCodeReason
«Union»
SequenceBuildingFeatureFunctionCodeReason
«attribute»
+ values: BuildingFeatureFunctionCodeList [1..*] {ordered}
+ reason: VoidValueReason
«codeList»
BuildingFeatureFunctionCodeList

Figure 1. NAS feature type Building with property featureFunction

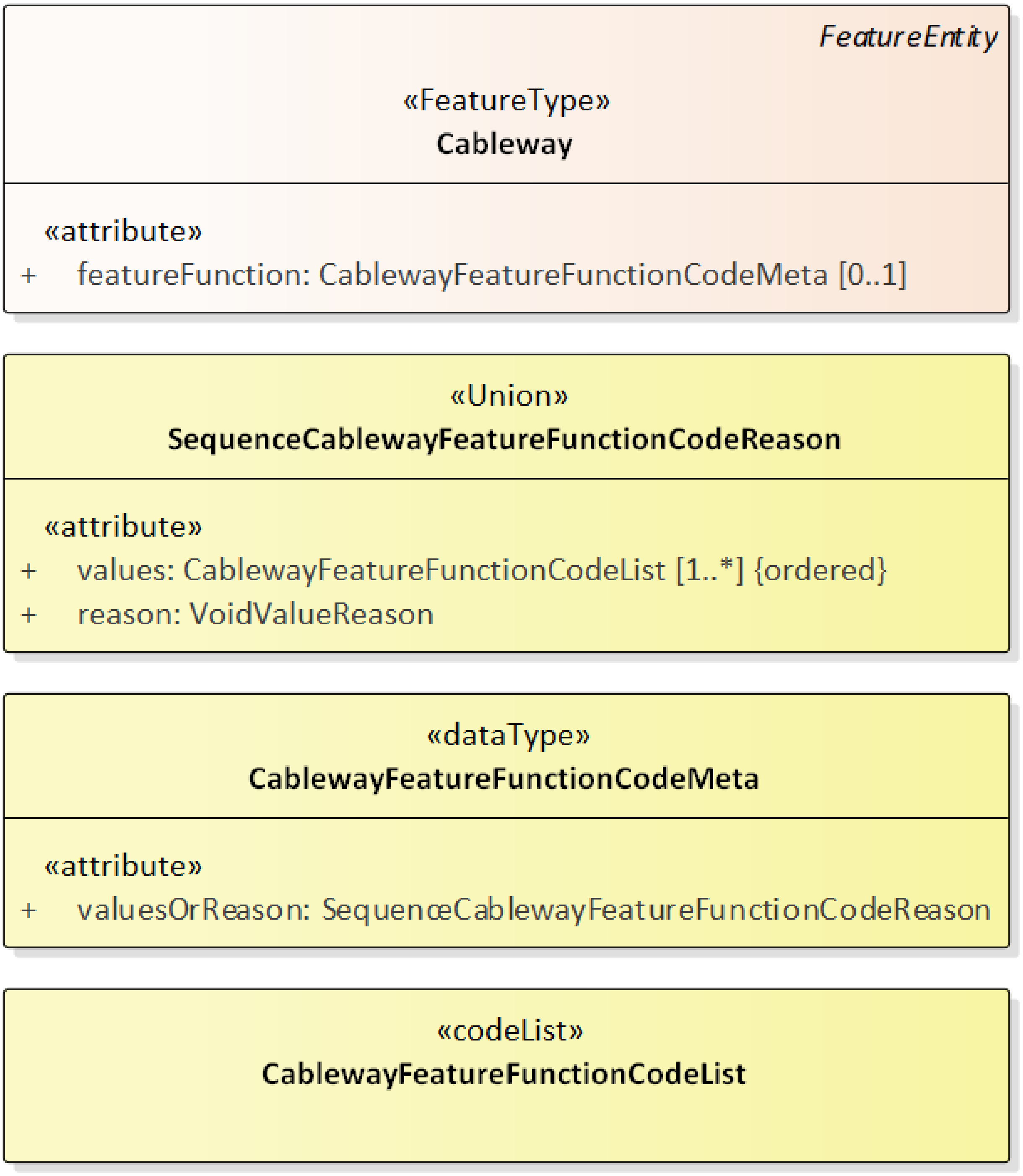
FeatureEntity
«FeatureType»
Cableway
«attribute»
+ featureFunction: CablewayFeatureFunctionCodeMeta [0..1]
«Union»
SequenceCablewayFeatureFunctionCodeReason
«attribute»
+ values: CablewayFeatureFunctionCodeList [1..*] {ordered}
+ reason: VoidValueReason
«dataType»
CablewayFeatureFunctionCodeMeta
«attribute»
+ valuesOrReason: SequenceCablewayFeatureFunctionCodeReason
«codeList»
CablewayFeatureFunctionCodeList

Figure 2. NAS feature type Cableway with property featureFunction

The specification of "featureFunction" is mostly identical in both cases:

- documentation: "Feature Function: A purpose of, or intended role served by, a feature."
- primaryCode: featureFunction
- secondaryCode: FFN

The value type is different. In Building, the value type is BuildingFeatureFunctionCodeMeta, while for Cableway it is CablewayFeatureFunctionCodeMeta. Following the links contained in the code lists to the NSG registry, it can be seen that the number of codes for BuildingFeatureFunctionCodeList [https://api.nsgreg.nga.mil/codelist/BuildingFeatureFunction] is significantly larger than those for CablewayFeatureFunctionCodeList [https://api.nsgreg.nga.mil/codelist/

CablewayFeatureFunction].

The use of feature type-specific code lists for property featureFunction supports precise modeling of the subset of feature function codes that apply specifically to a given feature type (like Building and Cableway). However, when deriving an ontology, this modeling approach results in locally scoped, unrelated featureFunction properties. Given that the specification of these properties is so similar, it would be helpful, from a knowledge point of view, to relate them. That can be achieved through the addition of general properties, which will be discussed in detail later on.

First, however, let us discuss the modeling approach and potential alternatives in more detail. If the featureFunction properties in the UML model used a common value type, i.e. a code list that contains all codes relevant for NAS feature types, the conversion to RDF could encode them through one global property. However, that would result in a less precise model and ontology, since there would be no feature type-specific restriction of the feature function codes. Adding OCL constraints to the UML model to define such a restriction is not an ideal solution either, since the feature function codes are managed outside of the UML model. That means that the set of feature function codes - in general, and for a specific feature type - can change at any time, and thus restrictions defined through OCL constraints would then become outdated.

General properties would also be useful to relate properties whose definition in UML is similar, but where the value type is not necessarily a code list. The value type could be, for example, an enumeration, or a basic type (Integer, Real, etc). In other words, the approach of adding general properties to an ontology is applicable to a wide range of UML properties.

7.2. Addition of General Properties

Ontologies can relate RDF/OWL properties through property axioms. Relating properties that have similar semantics can be achieved through generalizing properties. Such a general property would express the commonalities of the sub-properties. If the sub-property specifics (e.g. specific ranges) are of interest, then the sub-properties can be related to a general property through rdfs:subPropertyOf. Otherwise, the sub-properties can be mapped to the general property.

NOTE

- *Relating* an RDF/OWL property to another RDF/OWL property is achieved by adding an rdfs:subPropertyOf predicate to the ontology, with the first property as subject and the second property as object.
- *Mapping* means that a specific UML property is not converted to an individual RDF/OWL property. Instead, the UML property is implemented by another RDF/OWL property - which can be defined by another ontology, or be the result of converting another UML property of the application schema.

ShapeChange supports the conversion of a UML property from the application schema to a global RDF/OWL property, and mapping or relating other properties to it. The corresponding ShapeChange configuration elements are *RdfPropertyMapEntry* and *PropertyConversionParameter*. However, the global property could still define specifics (like a feature type-specific range or definition), and thus this approach is unsatisfying for relating properties with similar semantics. Therefore, ShapeChange has been extended to support the creation of new, general RDF/OWL properties.

Other properties that result from conversion of UML properties from the application schema can be mapped or related to these new properties via the aforementioned ShapeChange configuration elements *RdfPropertyMapEntry* and *PropertyConversionParameter*.

The configuration of a ShapeChange OWL target has been enhanced to include specifications for object and data properties (via the configuration element 'rdfGeneralProperties'). The following listing gives an example.

Listing 3. Example of an 'rdfGeneralProperties' ShapeChange configuration element

```xml
<rdfGeneralProperties>
 <GeneralObjectProperty>
  <name>accelerateStopDistAvail</name>
  <domainByUnionOfSubPropertyDomains>true</domainByUnionOfSubPropertyDomains>
  <equivalentProperty>neo-ent:landingDistanceAvailable</equivalentProperty>
  <disjointProperty>neo-ent:distanceValue</disjointProperty>
  <additionalProperty>
   <property>rdfs:label</property>
   <value lang="en">accelerateStopDistAvail</value>
  </additionalProperty>
  <additionalProperty>
   <property>skos:prefLabel</property>
   <value lang="en">Accelerate-Stop Distance Available</value>
  </additionalProperty>
  <additionalProperty>
   <property>skos:altLabel</property>
   <value lang="en">ASDA</value>
  </additionalProperty>
  <additionalProperty>
   <property>skos:definition</property>
   <value lang="en"><![CDATA[<omitted for brevity>]]></value>
  </additionalProperty>
  <inverseProperty>neo-ent:landingDistanceAvailable</inverseProperty>
  <inverseProperty>neo-ent:takeOffDistAvailable</inverseProperty>
  <propertyCharacteristicAxioms>InverseFunctional</propertyCharacteristicAxioms>
 </GeneralObjectProperty>
 <GeneralDataProperty>
  <name>consumableType</name>
  <domainByUnionOfSubPropertyDomains>true</domainByUnionOfSubPropertyDomains>
  <additionalProperty>
   <property>rdfs:label</property>
   <value lang="en">consumableType</value>
  </additionalProperty>
  <additionalProperty>
   <property>skos:prefLabel</property>
   <value lang="en">Consumable Type</value>
  </additionalProperty>
  <additionalProperty>
   <property>skos:definition</property>
   <value lang="en"><![CDATA[<omitted for brevity>]]></value>
  </additionalProperty>
  <isFunctional>true</isFunctional>
 </GeneralDataProperty>
</rdfGeneralProperties>
```

There are three options for defining the domain of a general property:

- The domain of the general property is explicitly defined by the configuration.

- The configuration contains the XML element *domainByUnionOfSubPropertyDomains* (with fixed value 'true'). Then the domain of the general property is constructed as owl:unionOf the domains of its sub-properties.
- Otherwise, the general property is created without domain assignment.

The range of a general property is either explicitly declared in the configuration, or the default applies, which is owl:Thing.

NOTE

- No specific conversion rule is necessary to instruct ShapeChange to generate general properties. The presence of the sc:rdfGeneralProperties configuration element is sufficient.
- As mentioned before, *RdfPropertyMapEntries* and *PropertyConversionParameters* can be used to either map UML properties from the application schema to general properties, or to relate them using rdfs:subPropertyOf. Through the enhancements introduced for Property Enrichment, subPropertyOf relationships can also be defined via tagged values on UML properties.
- ShapeChange does not check that the definition of a general property and the definitions of its subproperties are consistent. Contradicting axioms could be defined (e.g. through property enrichment) for the subproperties. Inconsistencies in the resulting ontology can be identified using ontology editors such as Protégé [https://protege.stanford.edu/].
- A ShapeChange transformation modifies a UML model in a certain way. Such transformations may result in relocation and renaming of UML properties, which can break certain modelling constructs, for example paths in OCL constraints and identification of UML properties in tagged values via fully qualified UML property names. In general, a ShapeChange transformation is NOT required to ensure that all model elements (OCL constraints, tagged values, etc.) are updated to prevent breaking any modelling constructs. The ShapeChange user needs to understand the transformations and targets that are part of the workflow.

The XML Schema definition of sc:rdfGeneralProperties (including documentation of elements and attributes) is provided in Annex A.

NOTE

- If no namespace is defined for a general property, then:
    - If target parameter *generalPropertyNamespaceAbbreviation* is defined, the property will be added to the namespace that is identified by the namespace abbreviation defined by the parameter. The target configuration must contain a namespace definition with a matching abbreviation.
    - Otherwise, the property will be added to the RDF namespace of the main ontology that is derived by ShapeChange.
- If a namespace is defined and it does not match the namespaces of one of the ontologies created by ShapeChange, or if no namespace is defined and the target parameter *generalPropertyNamespaceAbbreviation* is set, then ShapeChange will create a new ontology file to represent that namespace. In both cases, the general property belongs to a namespace that is configured by the target. If a location is defined for the namespace, the last segment of the path is used as the name of the ontology file. Otherwise, the file name is constructed by replacing all non-word characters of the namespace with underscores. That way, the name of the new ontology file should be unique.

Annex A: XML Schema Documents

This annex contains XML Schema definitions for ShapeChange extensions specified in OGC Testbed-14. The latest version of the configuration is available online at http://shapechange.net and https://github.com/ShapeChange/ShapeChange.

A.1. rdfGeneralProperties XSD

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

```
<element name="rdfGeneralProperties">
 <complexType>
  <sequence>
   <element maxOccurs="unbounded" minOccurs="0" ref="sc:AbstractRdfGeneralProperty"/>
  </sequence>
 </complexType>
</element>
<element abstract="true" name="AbstractRdfGeneralProperty" type=
"sc:AbstractRdfGeneralPropertyType"/>
<complexType abstract="true" name="AbstractRdfGeneralPropertyType">
 <sequence>
  <element minOccurs="0" name="namespaceAbbreviation" type="string">
   <annotation>
    <documentation>Abbreviation of the RDF namespace to which the general property
shall belong. Note: the target configuration must contain a namespace definition with
a matching abbreviation, or the abbreviation matches one of the ontologies that is
created by ShapeChange.</documentation>
   </annotation>
  </element>
  <element name="name" type="string">
   <annotation>
    <documentation>Local name of the property.</documentation>
   </annotation>
  </element>
  <choice minOccurs="0">
   <element maxOccurs="1" minOccurs="1" name="domain" type="string">
    <annotation>
     <documentation>The domain of the general property. The element value is the IRI
of an RDF/OWL class. Note: the value is expected to be given as a QName, with the
namespace prefix matching a namespace abbreviation of the namespaces declared in the
configuration.</documentation>
    </annotation>
   </element>
   <element fixed="true" name="domainByUnionOfSubPropertyDomains" type="boolean">
    <annotation>
     <documentation>Use this element to construct the domain of the general property
as the union of all domains of its sub-properties.</documentation>
```

```
    </annotation>
   </element>
  </choice>
  <element minOccurs="0" name="range" type="string">
   <annotation>
    <documentation>Range of the general 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>
  </element>
  <element maxOccurs="unbounded" minOccurs="0" name="equivalentProperty" type="string
">
   <annotation>
    <documentation>Identifies a UML or RDF/OWL property to which the RDF/OWL
implementation of the UML property is equivalent. Note: the value is expected to be
given as either 1) a QName, with the namespace prefix matching the namespace
abbreviation of a namespace declared in the configuration, or 2) the full name of a
UML property (i.e. the package qualified name of the UML property, starting with the
application schema package, and using '::' as separator).</documentation>
   </annotation>
  </element>
  <element maxOccurs="unbounded" minOccurs="0" name="disjointProperty" type="string">
   <annotation>
    <documentation>Identifies a UML or RDF/OWL property to which the RDF/OWL
implementation of the UML property is disjoint. Note: the value is expected to be
given as either 1) a QName, with the namespace prefix matching the namespace
abbreviation of a namespace declared in the configuration, or 2) the full name of a
UML property (i.e. the package qualified name of the UML property, starting with the
application schema package, and using '::' as separator).</documentation>
   </annotation>
  </element>
  <element maxOccurs="unbounded" minOccurs="0" name="subPropertyOf" type="string">
   <annotation>
    <documentation>Identifies a UML or RDF/OWL property for which the RDF/OWL
implementation of the UML property is a subPropertyOf. Note: the value is expected to
be given as either 1) a QName, with the namespace prefix matching the namespace
abbreviation of a namespace declared in the configuration, or 2) the full name of a
UML property (i.e. the package qualified name of the UML property, starting with the
application schema package, and using '::' as separator).</documentation>
   </annotation>
  </element>
  <element maxOccurs="unbounded" minOccurs="0" name="additionalProperty">
   <annotation>
    <documentation>Additional property expression that applies to the general property
(e.g. skos:definition, rdfs:label).</documentation>
   </annotation>
   <complexType>
    <sequence>
     <element name="property" type="string">
      <annotation>
       <documentation>Identifies the additional 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>
     </element>
     <element maxOccurs="unbounded" name="value">
      <annotation>
       <documentation>Value of the additional property. A simple type if the
additional property is a datatype property (@isIRI=false), otherwise an IRI
(isIRI=true). A simple value can be tagged with a language identifier (@lang).
</documentation>
      </annotation>
      <complexType>
       <simpleContent>
        <extension base="string">
         <attribute name="lang" type="string"/>
         <attribute default="false" name="isIRI" type="boolean">
          <annotation>
           <documentation>If set to true, the value is interpreted as an
IRI.</documentation>
          </annotation>
         </attribute>
        </extension>
       </simpleContent>
      </complexType>
     </element>
    </sequence>
   </complexType>
  </element>
 </sequence>
</complexType>
<element name="GeneralObjectProperty" substitutionGroup="
sc:AbstractRdfGeneralProperty"
 type="sc:GeneralObjectPropertyType"/>
<complexType name="GeneralObjectPropertyType">
 <complexContent>
  <extension base="sc:AbstractRdfGeneralPropertyType">
   <sequence>
    <element maxOccurs="unbounded" minOccurs="0" name="inverseProperty" type="string">
     <annotation>
      <documentation>Identifies a UML or RDF/OWL property to which the RDF/OWL
implementation of the UML property is inverse. Note: the value is expected to be given
as either 1) a QName, with the namespace prefix matching the namespace abbreviation of
a namespace declared in the configuration, or 2) the full name of a UML property (i.e.
the package qualified name of the UML property, starting with the application schema
package, and using '::' as separator).</documentation>
     </annotation>
    </element>
    <element default="" minOccurs="0" name="propertyCharacteristicAxioms"
     type="sc:ObjectPropertyCharacteristicAxiomList">
     <annotation>
      <documentation>List of characteristic axioms that apply to the general object
```

```
property.</documentation>
     </annotation>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<element name="GeneralDataProperty" substitutionGroup="sc:AbstractRdfGeneralProperty"
 type="sc:GeneralDataPropertyType"/>
<complexType name="GeneralDataPropertyType">
 <complexContent>
  <extension base="sc:AbstractRdfGeneralPropertyType">
   <sequence>
    <element default="false" minOccurs="0" name="isFunctional" type="boolean">
     <annotation>
      <documentation>If set to true, the general property is a functional data
property.</documentation>
     </annotation>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<simpleType name="ObjectPropertyCharacteristicAxiomList">
 <list itemType="sc:ObjectPropertyCharacteristicAxiom"/>
</simpleType>
<simpleType name="ObjectPropertyCharacteristicAxiom">
 <restriction base="string">
  <enumeration value="Functional"/>
  <enumeration value="InverseFunctional"/>
  <enumeration value="Reflexive"/>
  <enumeration value="Irreflexive"/>
  <enumeration value="Symmetric"/>
  <enumeration value="Asymmetric"/>
  <enumeration value="Transitive"/>
 </restriction>
</simpleType>
```

Annex B: Conversion of NAS OCL Constraints to OWL

The NAS contains numerous OCL constraints that define model restrictions and conditions. For OGC Testbed-14, the sponsor categorized all existing NAS OCL constraints. 54 different types of constraints have been identified. The following sections of this Annex document each type of constraint, giving one typical constraint in the NAS as an example, and then stating whether or not the given constraint type can be translated to OWL. For some of the constraint types that can be translated, an example of the corresponding OWL expression is given in a pseudo-OWL-functional-style-syntax. That syntax uses the following abbreviations to represent certain OWL element types:

- OPE(*a*) - identifies object property *a*
- DPE(*b*) - identifies data property *b*
- DR(*c*) - identifies data range *c*
- CE(*d*) - identifies class *d*
- IND(*e*) - identifies individual *e*

NOTE The OWL expression of an OCL constraint is defined as an OWL restriction on the OWL class that provides the context for the OCL constraint. The OWL expressions in pseudo-OWL-functional-style-syntax of the following subsections represent the restriction itself. How the restriction is attached to the class that provides the context of the OCL constraint is not shown. An example that shows how the restriction would be attached to the class is given in Attaching the resulting OWL expression to a class.

B.1. NAS OCL Constraints for which a translation to OWL is possible

B.1.1. Association class end case

B.1.1.1. OCL constraint

```
inv: self.oclIsTypeOf(EventParticipation) /* Placeholder NULL Constraint */
```

- Class in which the example constraint is defined: EventParticipation
- Constraint description: If the associated event participant is an actor entity, then their participation role is either agent, consumer, experiencer, initiator, instrumental, intentional agent, object of event, observant, official recognition, provider, reactionary manipulator, receiver, or witness.
- Additional note on the constraint: The ShapeChange OCL parser does not currently handle constraints that traverse the "ends" of Association Classes due to the expressive limitations of XML Schema and Schematron. An always-true OCL constraint is currently specified instead of

the following, correct, OCL constraint:

```
inv: (eventParticipant->oclIsKindOf(ActorEntity) and
participationRole.valueOrReason.value->notEmpty()) implies
(participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::agent or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::consumer or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::experiencer or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::initiator or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::instrumental or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::intentionalAgent or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::objectOfEvent or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::observant or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::officialRecognition or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::provider or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::reactionaryManipulator or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::receiver or
participationRole.valueOrReason.value =
EventParticipationParticipationRoleCodeList::witness)
```

B.1.1.2. Translation to OWL

A translation of the correct OCL constraint to an OWL class expression is possible.

Placeholder constraints should be deleted before conversion to OWL takes place, for example via a new ShapeChange transformation (which would have to be implemented). Otherwise the resulting ontology would have tautological class expression axioms (in this case: EventParticipant SubClassOf: EventParticipant).

If the placeholder constraint was replaced by the correct constraint, it can be realized with the following OWL class expression:

```
ObjectUnionOf(
  ObjectComplementOf(
    ObjectIntersectionOf (
      ObjectIntersectionOf (
        ObjectSomeValuesFrom ( OPE(eventParticipant) CE(ActorEntity) )
        ObjectAllValuesFrom ( OPE(eventParticipant) CE(ActorEntity) )
      )
      ObjectSomeValuesFrom ( OPE(participationRole)
        ObjectSomeValuesFrom ( OPE(valueOrReason)
          ObjectSomeValuesFrom ( OPE(value) CE(Codelist) ) ) )
    )
  )
  ObjectAllValuesFrom ( OPE(participationRole)
    ObjectAllValuesFrom ( OPE(valueOrReason)
      ObjectAllValuesFrom ( OPE(value)
        ObjectOneOf (
          IND (EventParticipationParticipationRoleCodeList::agent)
          IND (EventParticipationParticipationRoleCodeList::consumer)
          IND (EventParticipationParticipationRoleCodeList::experiencer)
          IND (EventParticipationParticipationRoleCodeList::initiator)
          IND (EventParticipationParticipationRoleCodeList::instrumental)
          IND (EventParticipationParticipationRoleCodeList::intentionalAgent)
          IND (EventParticipationParticipationRoleCodeList::objectOfEvent)
          IND (EventParticipationParticipationRoleCodeList::observant)
          IND (EventParticipationParticipationRoleCodeList::officialRecognition)
          IND (EventParticipationParticipationRoleCodeList::provider)
          IND (EventParticipationParticipationRoleCodeList::reactionaryManipulator)
          IND (EventParticipationParticipationRoleCodeList::receiver)
          IND (EventParticipationParticipationRoleCodeList::witness)
        )
      )
    )
  )
)
```

The OWL expression represents a translation of the following OCL constraint, whose expression has been modified so that the shorthand for collect notation ("participationRole.valueOrReason.value") is replaced with existential and universal quantifications, which implement the original intent. Note that the OWL expression also applies one of the optimizations discussed in Considerations for translation optimization.

```
inv: (eventParticipant->oclIsKindOf(ActorEntity) and participationRole-
>exists(x1|x1.valueOrReason->exists(x2|x2.value->notEmpty()))) implies
(participationRole->forAll(x1|x1.valueOrReason->forAll(x2|x2.value->forAll(x3|x3 =
EventParticipationParticipationRoleCodeList::agent or x3 =
EventParticipationParticipationRoleCodeList::consumer or x3 =
EventParticipationParticipationRoleCodeList::experiencer or x3 =
EventParticipationParticipationRoleCodeList::initiator or x3 =
EventParticipationParticipationRoleCodeList::instrumental or x3 =
EventParticipationParticipationRoleCodeList::intentionalAgent or x3 =
EventParticipationParticipationRoleCodeList::objectOfEvent or x3 =
EventParticipationParticipationRoleCodeList::observant or x3 =
EventParticipationParticipationRoleCodeList::officialRecognition or x3 =
EventParticipationParticipationRoleCodeList::provider or x3 =
EventParticipationParticipationRoleCodeList::reactionaryManipulator or x3 =
EventParticipationParticipationRoleCodeList::receiver or x3 =
EventParticipationParticipationRoleCodeList::witness))))
```

B.1.2. Entity Instances Disallowed

B.1.2.1. OCL constraint

```
inv: self->isEmpty()
```

- Class in which the example constraint is defined: CI_Organisation
- Constraint description: No instances of this resource party organisation (ISO TC211) are allowed.

B.1.2.2. Translation to OWL

A translation of the OCL constraint - as it is currently defined - to an OWL class expression is not possible. However, it is possible to represent the intent through new OCL constraints, which can be translated to OWL.

To understand the intent of the OCL constraint, an overview of the relevant classes from the conceptual model is needed (see Figure 3).

[ResourcePartyOrg] | *images/ResourcePartyOrg.png*

Figure 3. CI_Organisation and its NAS subtype ResourcePartyOrg

Since the constraint is defined on the non-abstract class CI_Organisation, the intent apparently is to essentially make CI_Organisation abstract, so that only instances of its NAS subtype ResourcePartyOrg exist.

The following approaches do not achieve this goal:

- The OCL constraint attempts to prevent any instances of CI_Organisation, without breaking the model. From a UML perspective, that means that the class (from ISO 19115-1) is made abstract.

However, OWL does not know the concept of abstract classes. The ISO 19150-2 encoding rule therefore only adds an annotation property (iso19150-2:isAbstract=true) to the OWL class that represents an abstract class. In general, an OWL reasoner would not flag instances of such OWL classes as inconsistencies. Therefore, translating this specific constraint by adding the annotation property iso19150-2:isAbstract=true does not achieve the intended behavior.

- Another angle to approach the translation of the constraint is to view CI_Organisation as being equivalent to owl:Nothing. However, then the class would not be satisfiable, which would likely cause consistency errors to be reported by a reasoner.
  Also note that an OCL constraint that is defined on a supertype also applies to its subtypes, which would then mean that ResourcePartyOrg is also equivalent to owl:Nothing.
- In general, the OCL constraint could be used to prevent the conversion of properties that have the class (in which the constraint is defined) as value type. That would only be feasible if the class has no subtypes, or if none of the subtypes is converted either. The latter is the case here, because, as mentioned before, the OCL constraint automatically applies to subtypes as well. If the class belongs to the application schema (instead of belonging to an external schema), then another consequence of the constraint could be to prevent the conversion of the UML class and its subtypes to OWL classes altogether. However, that does not reflect the intent.

One solution could be to use SHACL to identify if any RDF representation of CI_Organisation occurs in RDF data, and to report a validation error if that is the case. Since the conversion of OCL to SHACL was not investigated in OGC Testbed-14, further analysis (see Deriving SHACL with ShapeChange) is needed to confirm that using SHACL indeed is a solution.

The correct way to represent the intent - that only instances of ResourcePartyOrg are used as values of properties with the value type CI_Organisation - is to define OCL constraints for each of these properties, as follows: *inv: self.{propertyName}→oclIsKindOf(ResourcePartyOrg)*. For example, class CI_Individual illustrated in Figure 3 would have the OCL constraint: *inv: self.organisation→oclIsKindOf(ResourcePartyOrg)*. Such OCL constraints can be translated to OWL class expressions, which can be evaluated by a reasoner.

NOTE It would also be worth considering getting away from preventing instances of specific supertypes altogether. In some scenarios, it may be desirable to assert that a certain resource is of a more generic type, because that might be the best classification that can be made at that time. Later on, if additional information has become available, it may be asserted that the type of the resource is a particular subclass.

B.1.3. New case #1 (single value)

B.1.3.1. OCL constraint

```
inv: area.valueOrReason.value->notEmpty() implies
area.valueOrReason.value.oclIsTypeOf(MeasureNonNegative)
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: The Feature Area attribute value of the access zone is specified using

"MeasureNonNegative".

B.1.3.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectUnionOf(
  ObjectComplementOf(
    ObjectSomeValuesFrom ( OPE(area)
      ObjectSomeValuesFrom ( OPE(valueOrReason)
        ObjectSomeValuesFrom ( OPE(value) CE(Measure) ) ) )
  )
  ObjectAllValuesFrom ( OPE(area)
    ObjectAllValuesFrom ( OPE(valueOrReason)
      ObjectAllValuesFrom ( OPE(value) CE(MeasureNonNegative) )
    )
  )
)
```

The corresponding OCL constraint that does not use the shorthand for collect notation is:

```
inv: area->exists(x1|x1.valueOrReason->exists(x2|x2.value->notEmpty())) implies area-
>forAll(x1|x1.valueOrReason->forAll(x2|x2.value-
>forAll(x3|x3.oclIsTypeOf(MeasureNonNegative))))
```

B.1.4. New case #2 (multiple values)

B.1.4.1. OCL constraint

```
inv: airfieldSymbolType.valueOrReason.value->notEmpty() implies
airfieldSymbolType.valueOrReason.value-
>forAll(x|x.oclIsTypeOf(HeliportAirfieldSymbolType))
```

- Class in which the example constraint is defined: Heliport
- Constraint description: The Airfield Symbol Type attribute value of the Heliport must be in accordance with 'HeliportAirfieldSymbolType'.

B.1.4.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectUnionOf(
  ObjectComplementOf(
    ObjectSomeValuesFrom ( OPE(airfieldSymbolType)
      ObjectSomeValuesFrom ( OPE(valueOrReason)
        ObjectSomeValuesFrom ( OPE(value) CE(Enumeration) ) ) )
  )
  ObjectAllValuesFrom ( OPE(airfieldSymbolType)
    ObjectAllValuesFrom ( OPE(valueOrReason)
      ObjectAllValuesFrom ( OPE(value) CE(HeliportAirfieldSymbolType) )
    )
  )
)
```

The corresponding OCL constraint that does not use the shorthand for collect notation is:

```
inv: airfieldSymbolType->exists(x1|x1.valueOrReason->exists(x2|x2.value->notEmpty()))
implies airfieldSymbolType->forAll(x1|x1.valueOrReason->forAll(x2|x2.value-
>forAll(x3|x3.oclIsTypeOf(HeliportAirfieldSymbolType))))
```

B.1.5. Property Allowed Listed Value (multiple choice)

B.1.5.1. OCL constraint

```
inv: featureOperationalStatus.valueOrReason.value->notEmpty() implies
featureOperationalStatus.valueOrReason.value =
FacilityFeatureOperationalStatusType::nonOperational or
featureOperationalStatus.valueOrReason.value =
FacilityFeatureOperationalStatusType::operational or
featureOperationalStatus.valueOrReason.value =
FacilityFeatureOperationalStatusType::partiallyOperational or
featureOperationalStatus.valueOrReason.value =
FacilityFeatureOperationalStatusType::planned or
featureOperationalStatus.valueOrReason.value =
FacilityFeatureOperationalStatusType::temporarilyNonOperational
```

- Class in which the example constraint is defined: AerialFarm
- Constraint description: The Feature Operational Status of an aerial farm (inherited from [Facility]) is one of: Non-operational; Operational; Partially Operational; Planned; or Temporarily Non-operational.

B.1.5.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

The constraint should first be converted to:

```
inv: featureOperationalStatus->exists(x1|x1.valueOrReason->exists(x2|x2.value-
>notEmpty())) implies featureOperationalStatus->forAll(x1|x1.valueOrReason-
>forAll(x2|x2.value->forAll(x3|x3 =
FacilityFeatureOperationalStatusType::nonOperational or x3 =
FacilityFeatureOperationalStatusType::operational or x3 =
FacilityFeatureOperationalStatusType::partiallyOperational or x3 =
FacilityFeatureOperationalStatusType::planned or x3 =
FacilityFeatureOperationalStatusType::temporarilyNonOperational)
```

Then the constraint can be translated to an OWL expression similar to the one shown for the association class end case:

```
ObjectUnionOf(
  ObjectComplementOf(
    ObjectSomeValuesFrom ( OPE(featureOperationalStatus)
      ObjectSomeValuesFrom ( OPE(valueOrReason)
        ObjectSomeValuesFrom ( OPE(value) CE(Enumeration) ) ) )
  )
  ObjectAllValuesFrom ( OPE(featureOperationalStatus) (
    ObjectAllValuesFrom ( OPE(valueOrReason) (
      ObjectAllValuesFrom ( OPE(value) (
        ObjectOneOf (
          IND (FacilityFeatureOperationalStatusType::nonOperational)
          IND (FacilityFeatureOperationalStatusType::operational)
          IND (FacilityFeatureOperationalStatusType::partiallyOperational)
          IND (FacilityFeatureOperationalStatusType::planned)
          IND (FacilityFeatureOperationalStatusType::temporarilyNonOperational)
        )
      )
    )
  )
)
```

B.1.6. Property Allowed Listed Value (single specific)

B.1.6.1. OCL constraint

```
inv: cropInfo.cropSpecies.valuesOrReason.values->notEmpty() implies
(cropInfo.cropSpecies.valuesOrReason.values->size()=1 and
cropInfo.cropSpecies.valuesOrReason.values->forAll(e|e =
CropInfoCropSpeciesCodeList::hop))
```

- Class in which the example constraint is defined: HopField
- Constraint description: The Crop Species attribute value list of the hop field (from its associated [CropInfo]) contains exactly the single value 'hop'.

B.1.6.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

The constraint should first be converted to:

```
inv: cropInfo->exists(x1|x1.cropSpecies->exists(x2|x2.valuesOrReason-
>exists(x3|x3.values->notEmpty()))) implies (cropInfo->forAll(x1|x1.cropSpecies-
>forAll(x2|x2.valuesOrReason->forAll(x3|x3.values->size()=1 and x3.values->forAll(e|e
= CropInfoCropSpeciesCodeList::hop)))
```

Then the constraint can be translated to the following OWL expression:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(cropInfo)
      ObjectSomeValuesFrom ( OPE(cropSpecies)
        ObjectSomeValuesFrom ( OPE(valuesOrReason)
          ObjectSomeValuesFrom ( OPE(values) CE(Codelist) ) ) ) )
  )
  ObjectAllValuesFrom ( OPE(cropInfo)
    ObjectAllValuesFrom ( OPE(cropSpecies)
      ObjectAllValuesFrom ( OPE(valuesOrReason)
        ObjectIntersectionOf (
          ObjectExactCardinality ( 1 OPE(values) )
          ObjectAllValuesFrom ( OPE(values)
            ObjectOneOf (
              IND (CropInfoCropSpeciesCodeList::hop)
            )
          )
        )
      )
    )
  )
)
```

B.1.7. Property Allowed Listed Values (array)

B.1.7.1. OCL constraint

```
inv: featureFunction.valuesOrReason.values->notEmpty() implies
(featureFunction.valuesOrReason.values->forAll(e|(e =
FacilityFeatureFunctionCodeList::mainTelephoneExchange or e =
FacilityFeatureFunctionCodeList::radioBroadcasting or e =
FacilityFeatureFunctionCodeList::satelliteTelecom or e =
FacilityFeatureFunctionCodeList::scientificResearchDevel or e =
FacilityFeatureFunctionCodeList::signalling or e =
FacilityFeatureFunctionCodeList::telecommunications or e =
FacilityFeatureFunctionCodeList::televisionBroadcasting or e =
FacilityFeatureFunctionCodeList::wiredTelecom or e =
FacilityFeatureFunctionCodeList::wirelessTelecom)) and
featureFunction.valuesOrReason.values->isUnique(x|x))
```

- Class in which the example constraint is defined: AerialFarm
- Constraint description: The Feature Function of an aerial farm (inherited from [Facility]) is (without repetition) zero or more of: Main Telephone Exchange; Radio Broadcasting; Satellite Telecommunications; Scientific Research and Development; Signalling; Telecommunications; Television Broadcasting; Wired Telecommunications; and Wireless Telecommunications.

B.1.7.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible, if the constraint is revised.

As described in the translation chapter, isUnique cannot be translated. Since OWL does not know duplicate values, the isUnique-check of the constraint can be ignored. When ShapeChange encounters an iterator call isUnique(), it cannot by itself restructure the OCL constraint. The translation of the constraint would therefore be stopped and an error reported. The constraint would need to be transformed beforehand, for example replacing it with an expression that does not have the isUnique check.

If the constraint is converted to:

```
inv: featureFunction->exists(x1|x1.valuesOrReason->exists(x2|x2.values->notEmpty()))
implies featureFunction->forAll(x1|x1.valuesOrReason->forAll(x2|x2.values->forAll(e|e
= FacilityFeatureFunctionCodeList::mainTelephoneExchange or e =
FacilityFeatureFunctionCodeList::radioBroadcasting or e =
FacilityFeatureFunctionCodeList::satelliteTelecom or e =
FacilityFeatureFunctionCodeList::scientificResearchDevel or e =
FacilityFeatureFunctionCodeList::signalling or e =
FacilityFeatureFunctionCodeList::telecommunications or e =
FacilityFeatureFunctionCodeList::televisionBroadcasting or e =
FacilityFeatureFunctionCodeList::wiredTelecom or e =
FacilityFeatureFunctionCodeList::wirelessTelecom)))
```

Then the constraint can be translated to the following OWL expression:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(featureFunction)
      ObjectSomeValuesFrom ( OPE(valuesOrReason)
        ObjectSomeValuesFrom ( OPE(values) CE(Codelist) ) ) )
  )
  ObjectAllValuesFrom ( OPE(featureFunction)
    ObjectAllValuesFrom ( OPE(valuesOrReason)
      ObjectAllValuesFrom ( OPE(values)
        ObjectOneOf (
          IND (FacilityFeatureFunctionCodeList::mainTelephoneExchange)
          IND (FacilityFeatureFunctionCodeList::radioBroadcasting)
          IND (FacilityFeatureFunctionCodeList::satelliteTelecom)
          IND (FacilityFeatureFunctionCodeList::scientificResearchDevel)
          IND (FacilityFeatureFunctionCodeList::signalling)
          IND (FacilityFeatureFunctionCodeList::telecommunications)
          IND (FacilityFeatureFunctionCodeList::televisionBroadcasting)
          IND (FacilityFeatureFunctionCodeList::wiredTelecom)
          IND (FacilityFeatureFunctionCodeList::wirelessTelecom)
        )
      )
    )
  )
)
```

B.1.8. Property Co-constraint (conditional populated)

B.1.8.1. OCL constraint

```
inv: pavementClassNumber.valueOrReason.value->notEmpty() implies
(pavementTypeForPcnDeterm.valueOrReason.value->notEmpty() and
pcnPavementSubStrengthCat.valueOrReason.value->notEmpty() and
pcnMaxAllowTirePressCat.valueOrReason.value->notEmpty() and
pcnEvaluationMethod.valueOrReason.value->notEmpty())
```

- Class in which the example constraint is defined: AerodromePavementInfo
- Constraint description: If the Pavement Classification Number (PCN) attribute value of the aerodrome pavement information is present then the four related attributes (Pavement Type for PCN Determination, PCN Pavement Subgrade Strength Category, PCN Maximum Allowable Tire Pressure Category, and PCN Evaluation Method) must also have values.

B.1.8.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.9. Property Co-constraint (at least one)

B.1.9.1. OCL constraint

```
inv: hydroBaseRef->notEmpty() or hydroBaseRefName->notEmpty()
```

- Class in which the example constraint is defined: HydroBaseHeightRefLevel
- Constraint description: Either the reference datum or the reference datum name must be specified.

B.1.9.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectUnionOf(
 ObjectSomeValuesFrom ( OPE(hydroBaseRef) CE(HydroBaseReferenceDatumCodeList) )
 ObjectSomeValuesFrom ( OPE(hydroBaseRefName) CE(TextLexUnconstrained) )
)
```

B.1.10. Property Co-constraint (related entity)

B.1.10.1. OCL constraint

```
inv: waterResourceInfo->notEmpty() implies (pipelineType.valuesOrReason.values =
PipelinePipelineCodeList::intakePipe or pipelineType.valuesOrReason.values =
PipelinePipelineCodeList::transportPipe)
```

- Class in which the example constraint is defined: Pipeline
- Constraint description: When a pipeline has associated [WaterResourceInfo] then its Pipeline Type is either Intake Pipe or Transport Pipe.

B.1.10.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.11. Property Co-constraint (related entity, exact listed value)

B.1.11.1. OCL constraint

```
inv: arrestSysEnergyAbsorbType.valueOrReason.value->notEmpty() implies (place-
>exists(p|p.oclIsKindOf(SurfacePositionInfo)) implies
arrestSysEnergyAbsorbType.valueOrReason.value =
ArrestingSystemArrestSysEnergyAbsorbCodeList::engMatArrestSystem)
```

- Class in which the example constraint is defined: ArrestingSystem
- Constraint description: If the Arresting System has an associated geometry that is a surface representation then the Arresting System Energy Absorber Type value must be an Engineered Materials Arresting System (EMAS).

B.1.11.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

The constraint could be restructured, merging the first two conditions, as follows:

```
inv: (arrestSysEnergyAbsorbType.valueOrReason.value->notEmpty() and place-
>exists(p|p.oclIsKindOf(SurfacePositionInfo))) implies
arrestSysEnergyAbsorbType.valueOrReason.value =
ArrestingSystemArrestSysEnergyAbsorbCodeList::engMatArrestSystem
```

This has the same logical conclusion as the original constraint, providing slightly better readability (though that might be a matter of opinion).

One could go a step further and define the constraint as:

```
inv: not(arrestSysEnergyAbsorbType.valueOrReason.value->notEmpty()) or not(place-
>exists(p|p.oclIsKindOf(SurfacePositionInfo))) or
arrestSysEnergyAbsorbType.valueOrReason.value =
ArrestingSystemArrestSysEnergyAbsorbCodeList::engMatArrestSystem
```

This represents a replacement of "a implies b" with "not(a) or b", plus application of DeMorgan's law to the negated "a", which is actually a conjunct in the example.

With:

- a = arrestSysEnergyAbsorbType.valueOrReason.value→notEmpty()
- b = place→exists(p|p.oclIsKindOf(SurfacePositionInfo))
- c = arrestSysEnergyAbsorbType.valueOrReason.value = ArrestingSystemArrestSysEnergyAbsorbCodeList::engMatArrestSystem

Then:

- the original constraint would be: a implies (b implies c) <=> not(a) or (not(b) or c) <=> not(a) or not(b) or c
- the first alternative would be: (a and b) implies c <=> not(a and b) or c <=> not(a) or not(b) or c

Essentially, what this means is that a translation of OCL to OWL may modify logical expressions, with the goal of simplifying the resulting OWL expression.

Finally, the constraint should be revised to avoid the use of the OCL collect operation:

```
inv: not(arrestSysEnergyAbsorbType->exists(x1|x1.valueOrReason->exists(x2|x2.value-
>notEmpty()))) or not(place->exists(p|p.oclIsKindOf(SurfacePositionInfo))) or
arrestSysEnergyAbsorbType->forAll(x1|x1.valueOrReason->forAll(x2|x2.value =
ArrestingSystemArrestSysEnergyAbsorbCodeList::engMatArrestSystem))
```

That OCL constraint can be translated to the following OWL expression:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(arrestSysEnergyAbsorbType)
      ObjectSomeValuesFrom ( OPE(valueOrReason)
        ObjectSomeValuesFrom ( OPE(value) CE(Codelist) ) ) )
  )
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(place) CE(SurfacePositionInfo) )
  )
  ObjectAllValuesFrom ( OPE(arrestSysEnergyAbsorbType)
    ObjectAllValuesFrom ( OPE(valueOrReason)
      ObjectAllValuesFrom ( OPE(value)
        ObjectOneOf (
          IND (ArrestingSystemArrestSysEnergyAbsorbCodeList::engMatArrestSystem)
        )
      )
    )
  )
)
```

B.1.12. Property Co-constraint (related entity, minimum listed value)

B.1.12.1. OCL constraint

```
inv: religiousSignificance->notEmpty() implies (featureFunction.valuesOrReason.values-
>notEmpty() implies featureFunction.valuesOrReason.values-
>exists(e|e=BuildingFeatureFunctionCodeList::religiousActivities))
```

- Class in which the example constraint is defined: Building
- Constraint description: If the building has religious significance (which is inherited from [FeatureEntity]), then the set of Feature Function values must include the value: Religious Activities.

B.1.12.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

A similar simplification as described for the previous section could be applied here. Then the OWL expression would be:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(religiousSignificance) CE(Religion) )
  )
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(featureFunction)
      ObjectSomeValuesFrom ( OPE(valuesOrReason)
        ObjectSomeValuesFrom ( OPE(values) CE(Codelist) ) ) )
  )
  ObjectAllValuesFrom ( OPE(featureFunction)
    ObjectAllValuesFrom ( OPE(valuesOrReason)
      ObjectSomeValuesFrom ( OPE(values)
        ObjectOneOf (
          IND (BuildingFeatureFunctionCodeList::religiousActivities)
        )
      )
    )
  )
)
```

B.1.13. Property Complex Type (disallowed listed value)

B.1.13.1. OCL constraint

```
inv: externalEntityIdentifier->forAll(x|x.valueOrReason.value.codeSpace <>
IdentifierNamespaceCodeList::basicEncyclopedia)
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: The External Entity Identifier of an access zone (inherited from [FeatureEntity]) shall not include a Basic Encyclopedia (BE) Number.

B.1.13.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

If the OCL constraint is revised to:

```
inv: externalEntityIdentifier->forAll(x1|x1.valueOrReason->forAll(x2|x2.value-
>forAll(x3|x3.codeSpace <> IdentifierNamespaceCodeList::basicEncyclopedia)))
```

Then the translation to an OWL expression would be:

```
ObjectAllValuesFrom ( OPE(externalEntityIdentifier)
  ObjectAllValuesFrom ( OPE(valueOrReason)
    ObjectAllValuesFrom ( OPE(value)
      ObjectAllValuesFrom ( OPE(codeSpace)
        ObjectComplementOf (
          ObjectOneOf (
            IND (IdentifierNamespaceCodeList::basicEncyclopedia)
          )
        )
      )
    )
  )
)
```

B.1.14. Property Complex Type (required listed value)

B.1.14.1. OCL constraint

```
inv: beliefSystemIdentifier.valueOrReason.value.codeSpace->notEmpty() implies
beliefSystemIdentifier.valueOrReason.value.codeSpace =
IdentifierNamespaceCodeList::beliefSystem
```

- Class in which the example constraint is defined: BeliefSystem
- Constraint description: The value of the Codespace (namespace) of a Belief System Identifier, if populated, is always 'beliefSystem'.

B.1.14.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.15. Property Complex Type (well-formed)

B.1.15.1. OCL constraint

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

- Class in which the example constraint is defined: Backshore
- Constraint description: The Maritime Bottom Characteristic attribute value triples of the backshore must be well-formed, where each attribute value is either specified or the reason why not is specified (and other than 'valueSpecified').

B.1.15.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.16. Property Excluded (assoc-role)

B.1.16.1. OCL constraint

```
inv: religiousSignificance->isEmpty()
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: An access zone never has religious significance (which is otherwise inherited from [FeatureEntity]).

B.1.16.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectExactCardinality( 0 OPE(religiousSignificance) )
```

B.1.17. Property Excluded (attribute)

B.1.17.1. OCL constraint

```
inv: locatedUnderground->isEmpty()
```

- Class in which the example constraint is defined: AerialFarm
- Constraint description: The Located Underground attribute of an aerial farm (inherited from [Facility]) is disallowed.

B.1.17.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectExactCardinality( 0 OPE(locatedUnderground) )
```

B.1.18. Property Numeric Range

B.1.18.1. OCL constraint

```
inv: bearingOfObject.valueOrReason.value->notEmpty() implies
((bearingOfObject.valueOrReason.value >= 0) and (bearingOfObject.valueOrReason.value
<= 359.9))
```

- Class in which the example constraint is defined: AdministrativeBoundary
- Constraint description: The Bearing of Object attribute value of the administrative boundary is restricted to fall in the range of 0 to 359.9, inclusive.

B.1.18.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

With one of the optimizations described in Considerations for translation optimization, the OWL expression would be:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(bearingOfObject)
      ObjectSomeValuesFrom ( OPE(valueOrReason)
        DataSomeValuesFrom ( DPE(value) DR(Measure) ) ) )
  )
  ObjectAllValuesFrom ( OPE(bearingOfObject)
    ObjectAllValuesFrom ( OPE(valueOrReason)
      DataAllValuesFrom ( DPE(value) DatatypeRestriction( DR(Measure) xsd:minInclusive
0 xsd:maxInclusive 359.9 ) )
    )
  )
)
```

B.1.19. Property Numeric Range (array)

B.1.19.1. OCL constraint

```
inv: baseElevation.valuesOrReason.values->notEmpty() implies
baseElevation.valuesOrReason.values->forAll(x|x.value->notEmpty() implies ((x.value >=
-400) and (x.value <= 30000)))
```

- Class in which the example constraint is defined: Aerial
- Constraint description: The Base Elevation attribute value list of the aerial contains elevation values that are restricted to fall in the range of -400 to 30000, inclusive.

B.1.19.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.20. Property Numeric Range (conjunct)

B.1.20.1. OCL constraint

```
inv: elevationAngle.valueOrReason.value->notEmpty() implies
((elevationAngle.valueOrReason.value >= -90) and (elevationAngle.valueOrReason.value
<= 90))
```

- Class in which the example constraint is defined: Aerial
- Constraint description: The Elevation Angle attribute value of the Aerial is restricted to fall in the range of -90 to 90, inclusive.

B.1.20.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.21. Property Required (conditional on related entity, exact listed value)

B.1.21.1. OCL constraint

```
inv: waterResourceInfo->notEmpty() implies (buriedUtilityType.valuesOrReason.values =
BuriedUtilityBuriedUtilCodeList::water)
```

- Class in which the example constraint is defined: BuriedUtility
- Constraint description: When a buried utility has associated [WaterResourceInfo] then its Buried Utility Type is always Water.

B.1.21.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible. However, the use of shorthand for collect in the OCL expression needs to be clarified and should be avoided (revise the constraint so that shorthand for collect is replaced, for example by forAll()).

B.1.22. Property Required (conditional, exact listed value, no meta)

B.1.22.1. OCL constraint

```
inv: referenceSystemType->notEmpty() implies referenceSystemType =
MD_ReferenceSystemTypeCode::geographicIdentifier
```

- Class in which the example constraint is defined: SpatRefSystGeoIdentifiers
- Constraint description: The Reference System Type of a Spatial Reference System Using Geographic Identifiers (ISO TC211) (inherited from [MD_ReferenceSystem]) is Geographic Identifier.

B.1.22.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.23. Property Required (minimum listed value)

B.1.23.1. OCL constraint

```
inv: featureFunction->notEmpty() implies featureFunction.valuesOrReason.values-
>exists(e|e=InstallationFeatureFunctionCodeList::defenceActivities)
```

- Class in which the example constraint is defined: MilitaryInstallation
- Constraint description: The Feature Function of a military installation (inherited from [Installation]) when populated must include Defence Activities.

B.1.23.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible. However, the use of shorthand for collect in the OCL expression needs to be clarified and should be avoided (revise the constraint so that shorthand for collect is replaced).

B.1.24. Property Restricted Value

B.1.24.1. OCL constraint

```
inv: tideInfluenced.valueOrReason.value->notEmpty() implies
tideInfluenced.valueOrReason.value = false
```

- Class in which the example constraint is defined: Bog
- Constraint description: The value of Tide Influenced of a bog (inherited from [Wetland]) is always FALSE.

B.1.24.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

If the constraint is converted to:

```
inv: tideInfluenced->exists(x1|x1.valueOrReason->exists(x2|x2.value->notEmpty()))
implies tideInfluenced->forAll(x1|x1.valueOrReason->forAll(x2|x2.value = false))
```

Then the constraint can be translated to the following OWL expression:

```
ObjectUnionOf (
  ObjectComplementOf (
    ObjectSomeValuesFrom ( OPE(tideInfluenced)
      ObjectSomeValuesFrom ( OPE(valueOrReason)
        DataSomeValuesFrom ( DPE(value) DR(Boolean) ) ) )
  )
  ObjectAllValuesFrom ( OPE(tideInfluenced)
    ObjectAllValuesFrom ( OPE(valueOrReason)
      DataAllValuesFrom ( DPE(value) "false"^^xsd:boolean )
    )
  )
)
```

B.1.25. Property String Length (equal)

B.1.25.1. OCL constraint

```
inv: igbUniqueStationCode.valueOrReason.value->notEmpty() implies
igbUniqueStationCode.valueOrReason.value.size() = 6
```

- Class in which the example constraint is defined: SurveyPoint
- Constraint description: The length of the International Gravimetric Bureau (IGB) Unique Station Code attribute value of the survey point is 6 characters.

B.1.25.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.26. Property String Length (maximum)

B.1.26.1. OCL constraint

```
inv: oceanographicSurveyIdent.valueOrReason.value->notEmpty() implies
oceanographicSurveyIdent.valueOrReason.value.size() <= 20
```

- Class in which the example constraint is defined: AcousticStation
- Constraint description: The length of the Oceanographic Survey Identifier attribute value of the acoustic station is no more than 20 characters.

B.1.26.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.27. Property String Length (complex array)

B.1.27.1. OCL constraint

```
inv: routeIdentification.valuesOrReason.values->notEmpty() implies
routeIdentification.valuesOrReason.values->forAll(x|x.routeDesignation->notEmpty()
implies (x.routeDesignation.size() <= 24))
```

- Class in which the example constraint is defined: FerryCrossing
- Constraint description: The length of the Route Designation value of the Route Identification attribute of the ferry crossing does not exceed 24 characters.

B.1.27.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.28. Property String Length (simple array)

B.1.28.1. OCL constraint

```
inv: routeDesignation.valuesOrReason.values->notEmpty() implies
routeDesignation.valuesOrReason.values->forAll(x|x.size() <= 24)
```

- Class in which the example constraint is defined: TankTrail
- Constraint description: The length of the Route Designation attribute value(s) of the tank trail do not exceed 24 characters.

B.1.28.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.29. Property Value Metadata

B.1.29.1. OCL constraint

```
inv: (area.valueOrReason.value->isEmpty() implies (area.valueOrReason.reason-
>notEmpty() and area.valueOrReason.reason <> VoidNumValueReason::valueSpecified)) and
(area.valueOrReason.value->notEmpty() implies area.valueOrReason.reason =
VoidNumValueReason::valueSpecified)
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: If the Area attribute value of the access zone is empty, then the reason why must be specified (and other than 'valueSpecified'); if the value is specified, then the reason why must be 'valueSpecified'.

B.1.29.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.30. Property Values Count

B.1.30.1. OCL constraint

```
inv: structMatType.valuesOrReason.values->notEmpty() implies
structMatType.valuesOrReason.values->size() <= 2
```

- Class in which the example constraint is defined: FloatingDryDock
- Constraint description: The Structural Material Type attribute value list of the floating dry dock contains at most two values.

B.1.30.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible. However, the constraint needs to be revised to avoid the use of shorthand for collect notation (for further details, see row #2 of table 2)

B.1.31. Property Values Metadata

B.1.31.1. OCL constraint

```
inv: (structMatType.valuesOrReason.values->isEmpty() implies
(structMatType.valuesOrReason.reason->notEmpty() and
structMatType.valuesOrReason.reason <> VoidValueReason::valueSpecified)) and
(structMatType.valuesOrReason.values->notEmpty() implies
structMatType.valuesOrReason.reason = VoidValueReason::valueSpecified)
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: If the Structural Material Type attribute value list of the access zone is empty then the reason why must be specified (and other than 'valueSpecified'); if the value is specified then the reason why must be 'valueSpecified'.

B.1.31.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.32. Related Entity Property Excluded (multiple)

B.1.32.1. OCL constraint

```
inv: declaredDistInfo.accelerateStopDistAvail->isEmpty() and
declaredDistInfo.landingDistanceAvailable->isEmpty() and
declaredDistInfo.takeOffDistAvailable->isEmpty() and
declaredDistInfo.takeOffRunAvailable->isEmpty()
```

- Class in which the example constraint is defined: FinalApproachTakeOffArea
- Constraint description: If the Final Approach and Take-off Area has an associated Runway Declared Distance Information, then none of the four attributes [Accelerate-Stop Distance Available, Landing Distance Available (LDA), Take-Off Distance Available (TODA), Take-Off Run Available (TORA)] are applicable.

B.1.32.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.33. Related Entity Property Excluded (single)

B.1.33.1. OCL constraint

```
inv: waterResourceInfo.hydrologicPersistence->isEmpty()
```

- Class in which the example constraint is defined: Aquifer
- Constraint description: The Hydrologic Persistence of an aquifer (from its associated [WaterResourceInfo]) is disallowed.

B.1.33.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.34. Related Entity Property Required

B.1.34.1. OCL constraint

```
inv: note->exists(x|x.memorandum->notEmpty())
```

- Class in which the example constraint is defined: Entity
- Constraint description: The Memorandum attribute of an associated Note entity must be populated (for example: with descriptive text where adequate attributes are not available to further describe the entity at a location).

B.1.34.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.35. Related Entity Property Value (conditional, restricted)

B.1.35.1. OCL constraint

```
inv: waterResourceInfo.waterUse.valuesOrReason.values->notEmpty() implies
(waterResourceInfo.waterUse.valuesOrReason.values =
WaterResourceInfoWaterUseCodeList::municipal)
```

- Class in which the example constraint is defined: FireHydrant
- Constraint description: The Water Use of a fire hydrant (from its associated [WaterResourceInfo]) is always Municipal.

B.1.35.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.36. Related Entity Property Value (conditional, restricted, array)

B.1.36.1. OCL constraint

```
inv: cropInfo.cropSpecies.valuesOrReason.values->notEmpty() implies
(cropInfo.cropSpecies.valuesOrReason.values->forAll(e|(e =
CropInfoCropSpeciesCodeList::barley or e = CropInfoCropSpeciesCodeList::cannabis or e
= CropInfoCropSpeciesCodeList::carob or e = CropInfoCropSpeciesCodeList::coffee or e =
CropInfoCropSpeciesCodeList::corkOak or e = CropInfoCropSpeciesCodeList::cotton or e =
CropInfoCropSpeciesCodeList::dryCrop or e = CropInfoCropSpeciesCodeList::fibreCrop or
e = CropInfoCropSpeciesCodeList::maize or e = CropInfoCropSpeciesCodeList::millet or e
= CropInfoCropSpeciesCodeList::oat or e = CropInfoCropSpeciesCodeList::oilCrop or e =
CropInfoCropSpeciesCodeList::opiumPoppy or e =
CropInfoCropSpeciesCodeList::ornamentalCrop or e = CropInfoCropSpeciesCodeList::peanut
or e = CropInfoCropSpeciesCodeList::potato or e = CropInfoCropSpeciesCodeList::pulse
or e = CropInfoCropSpeciesCodeList::rye or e = CropInfoCropSpeciesCodeList::sisal or e
= CropInfoCropSpeciesCodeList::sorghum or e = CropInfoCropSpeciesCodeList::sugarCrop
or e = CropInfoCropSpeciesCodeList::tobacco or e = CropInfoCropSpeciesCodeList::tuber
or e = CropInfoCropSpeciesCodeList::vegetableCrop or e =
CropInfoCropSpeciesCodeList::wheat)) and cropInfo.cropSpecies.valuesOrReason.values-
>isUnique(x|x))
```

- Class in which the example constraint is defined: CropLand
- Constraint description: A Crop Land never has a Crop Species (an attribute of the associated Crop Information) where the crop corresponds to that appropriate to an alternative feature; such as a: Hop Field; Rice Field or Vineyard. Therefore, its allowed values of Crop Species are zero or more of: Barley; Cannabis; Carob; Coffee; Cork Oak; Cotton; Dry Crop; Fibre Crop; Maize; Millet; Oat; Oil Crop; Opium Poppy; Ornamental Crop; Peanut; Potato; Pulse; Rye; Sisal; Sorghum; Sugar Crop; Tobacco; Tuber; Vegetable Crop; and Wheat.

B.1.36.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible - except that isUnique cannot be translated. Section Property Allowed Listed Values (array) has a similar example, also showing the translation to OWL.

B.1.37. Related Entity Property Value (guard, conditional, restricted)

B.1.37.1. OCL constraint

```
inv: (basicAdminUnitName->notEmpty() and
basicAdminUnitName.nameType.valueOrReason.value->notEmpty()) implies
(basicAdminUnitName.nameType.valueOrReason.value =
NameInfoNameCodeList::basicAdminUnitName)
```

- Class in which the example constraint is defined: BasicAdministrativeUnit
- Constraint description: When a basic administrative unit has associated [NameInfo] then its Name Type is a Basic Administrative Unit Name.

B.1.37.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.38. Related Entity Property Value (specific)

B.1.38.1. OCL constraint

```
inv: waterResourceInfo.hydrologicPersistence.valueOrReason.value->notEmpty() implies
waterResourceInfo.hydrologicPersistence.valueOrReason.value =
WaterResourceInfoHydrologicPersistenceType::perennial
```

- Class in which the example constraint is defined: Bog
- Constraint description: The Hydrologic Persistence of a bog (from its associated [WaterResourceInfo]) is always Perennial.

B.1.38.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.39. Related Entity Required

B.1.39.1. OCL constraint

```
inv: member->notEmpty()
```

- Class in which the example constraint is defined: EventCollection
- Constraint description: The Member association role of the Event Collection must be populated (empty Event Collections are not allowed).

B.1.39.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.40. Related Entity Required (at least one)

B.1.40.1. OCL constraint

```
inv: cropLand->notEmpty() or farm->notEmpty() or hopField->notEmpty() or
industrialFarm->notEmpty() or orchard->notEmpty() or plantNursery->notEmpty() or
ranch->notEmpty() or riceField->notEmpty() or vineyard->notEmpty()
```

- Class in which the example constraint is defined: CropInfo
- Constraint description: There exists at least one of the following associated entities: Crop Land, Farm, Hop Field, Industrial Farm, Orchard, Plant Nursery, Ranch, Rice Field, or Vineyard.

B.1.40.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.1.41. Related Entity Type (specific)

B.1.41.1. OCL constraint

```
inv: movementArea->exists(x|x.oclIsTypeOf(Helipad))
```

- Class in which the example constraint is defined: Heliport
- Constraint description: The heliport shall always have at least one associated helipad (which may possibly be present with any [Aerodrome] subclass).

B.1.41.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectSomeValuesFrom ( OPE(movementArea) CE(Helipad) )
```

B.1.42. Related Entity Type (restricted)

B.1.42.1. OCL constraint

```
inv: time->forAll(t|t.oclIsKindOf(TimePointInfo))
```

- Class in which the example constraint is defined: AntiShippingActivity
- Constraint description: The associated time representation(s) are point(s) representing the temporal instant of this anti-shipping activity.

B.1.42.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectAllValuesFrom ( OPE(time) CE(TimePointInfo) )
```

B.1.43. Related Entity Type Disallowed (place)

B.1.43.1. OCL constraint

```
inv: place->forAll(p| not(p.oclIsKindOf(PhysicalAddressInfo)))
```

- Class in which the example constraint is defined: AcousticSensor

- Constraint description: If there are associated place representations for this acoustic sensor, then they are not physical address(es).

B.1.43.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectAllValuesFrom ( OPE(place)
  ObjectComplementOf( CE(PhysicalAddressInfo) )
)
```

B.1.44. Related Entity Types Disallowed (place, conjunct)

B.1.44.1. OCL constraint

```
inv: place->forAll(p| not(p.oclIsKindOf(PhysicalAddressInfo)) and
not(p.oclIsKindOf(CurvePositionInfo)))
```

- Class in which the example constraint is defined: AerationBasin
- Constraint description: If there are associated place representations for this aeration basin, then they are neither physical address(es), nor positions established as curve(s).

B.1.44.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectAllValuesFrom ( OPE(place)
  ObjectIntersectionOf (
    ObjectComplementOf ( CE(PhysicalAddressInfo) )
    ObjectComplementOf ( CE(CurvePositionInfo) )
  )
)
```

B.1.45. Related Entity Types Disallowed (place, conjunct, complex)

B.1.45.1. OCL constraint

```
inv: place->forAll(p| not(p.oclIsKindOf(PhysicalAddressInfo)) and
not(p.oclIsKindOf(PointPositionInfo) or p.oclIsKindOf(CurvePositionInfo)))
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: If there are associated place representations for this access zone, then they are neither physical address(es), nor positions established as either point(s) or curve(s).

B.1.45.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectAllValuesFrom ( OPE(place)
  ObjectIntersectionOf (
    ObjectComplementOf ( CE(PhysicalAddressInfo) )
    ObjectComplementOf (
      ObjectUnionOf ( CE(PointPositionInfo) CE(CurvePositionInfo) )
    )
  )
)
```

B.1.46. Related Entity Types Disallowed (place, disjunct)

B.1.46.1. OCL constraint

```
inv: place->forAll(p| not(p.oclIsKindOf(PointPositionInfo) or
p.oclIsKindOf(CurvePositionInfo)))
```

- Class in which the example constraint is defined: AntiAircraftArtillerySite
- Constraint description: If there are associated place representations for this anti-aircraft artillery site, then they are not positions established as either point(s) or curve(s).

B.1.46.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

```
ObjectAllValuesFrom ( OPE(place)
  ObjectComplementOf (
    ObjectUnionOf ( CE(PointPositionInfo) CE(CurvePositionInfo) )
  )
)
```

B.1.47. Related Entity, Related Entity Property Required

B.1.47.1. OCL constraint

```
inv: geoNameCollection->notEmpty() implies (geoNameCollection.memberGeoName-
>exists(x|x.fullName->notEmpty()))
```

- Class in which the example constraint is defined: NamedLocation
- Constraint description: The Full Name attribute of an associated Geographic Name Information entity of the associated Geographic Name Collection must be populated.

B.1.47.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is possible.

B.2. NAS OCL Constraints for which a translation to OWL is not possible

B.2.1. Property Co-constraint (related entity, numeric comparison)

B.2.1.1. OCL constraint

```
inv: (width.valueOrReason.value->notEmpty() and
runwayDirection.runway.width.valueOrReason.value->notEmpty()) implies
((width.valueOrReason.value.lowerValue >=
runwayDirection.runway.width.valueOrReason.value.lowerValue) or
(width.valueOrReason.value.upperValue >=
runwayDirection.runway.width.valueOrReason.value.upperValue))
```

- Class in which the example constraint is defined: Stopway
- Constraint description: The Width attribute interval value of a stopway must equal or exceed the width interval value of its associated runway.

B.2.1.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is not possible.

A translation of this constraint would require variables within the OWL expression, to perform the comparisons for lower and upper values of width and runwayDirection.runway.width.

B.2.2. Property Co-constraint (type conditional, numeric comparison)

B.2.2.1. OCL constraint

```
inv: (self.oclIsTypeOf(Runway) and self.length.valueOrReason.value->notEmpty() and
self.width.valueOrReason.value->notEmpty()) implies
((self.length.valueOrReason.value.lowerValue >
self.width.valueOrReason.value.lowerValue) or
(self.length.valueOrReason.value.upperValue >
self.width.valueOrReason.value.upperValue))
```

- Class in which the example constraint is defined: AerodromeMoveArea
- Constraint description: The Length attribute interval value of a runway must exceed its width interval value.

B.2.2.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is not possible.

The same explanation as for Property Co-constraint (related entity, numeric comparison) applies.

B.2.3. Property Valid Numeric Interval

B.2.3.1. OCL constraint

```
inv: length.valueOrReason.value->notEmpty() implies
(length.valueOrReason.value.lowerValue <= length.valueOrReason.value.upperValue) and
((length.valueOrReason.value.lowerValue = length.valueOrReason.value.upperValue)
implies (length.valueOrReason.value.intervalClosureType =
IntervalClosureType::closedInterval)) and
((length.valueOrReason.value.intervalClosureType = IntervalClosureType::gtSemiInterval
or length.valueOrReason.value.intervalClosureType =
IntervalClosureType::gteSemiInterval) implies (length.valueOrReason.value.upperValue-
>isEmpty())) and ((length.valueOrReason.value.intervalClosureType =
IntervalClosureType::ltSemiInterval or length.valueOrReason.value.intervalClosureType
= IntervalClosureType::lteSemiInterval) implies
(length.valueOrReason.value.lowerValue->isEmpty()))
```

- Class in which the example constraint is defined: AccessZone
- Constraint description: The Length attribute value of the access zone must be a well-formed and numerically valid interval.

B.2.3.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is not possible.

The same explanation as for Property Co-constraint (related entity, numeric comparison) applies: A translation of the constraint would require variables within the OWL expression, to perform the comparisons for lower and upper values.

NOTE The constraint could be revised to remove the conditions that require numeric comparison of variables. On first glance, that would be similar to the constraint Property Allowed Listed Values (array), which can be revised to omit the unsupported isUnique() call. However, the situation is different: if the numeric comparisons were omitted here, crucial parts of the constraint would be missing. Additional mechanisms or languages for encoding the constraint, so that it can be checked on RDF encoded data, should be investigated – like SHACL (also see the related future work item).

B.2.4. Related Entity Property Numeric Range Restriction

B.2.4.1. OCL constraint

```
inv: movementArea->notEmpty() implies movementArea-
>forAll(x|((x.highestElevation.valuesOrReason.values->forAll(e|((e.value >=
self.aerodromeElevation.valueOrReason.value - 60.96) and (e.value <=
self.aerodromeElevation.valueOrReason.value + 60.96))))))
```

- Class in which the example constraint is defined: Aerodrome
- Constraint description: The Highest Elevation(s) of all aerodrome movement areas at an aerodrome must not differ by more than 200 feet from the Aerodrome Elevation.

B.2.4.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is not possible.

OWL does not support variables and addition or subtraction of literals.

B.2.5. Special case #1

B.2.5.1. OCL constraint

```
inv: self.oclIsTypeOf(TimePointInfo) /* Placeholder NULL Constraint */
```

- Class in which the example constraint is defined: TimePointInfo
- Constraint description: The Temporal Position attribute value of the time point information uses a Temporal Reference Frame that is specified using "http://api.nsgreg.nga.mil/codelist/CalendarSystem".
- Additional note on the constraint: The ShapeChange OCL parser is currently limited by the expressiveness of Schematron – which does not directly support pattern-based constraints; either a Java extension function with XPath 1.0 or the "matches" function in XPath 2.0 is required. An always-true OCL constraint is currently specified instead of the applicable pattern-based constraint.

B.2.5.2. Translation to OWL

It is unclear if the intent - as explained in the constraint description - can be represented in OWL. At the moment, the OCL constraint given as example is a placeholder. A real OCL constraint that expresses the intent needs to be given in order to perform the analysis.

B.2.6. Special case #2

B.2.6.1. OCL constraint

```
inv: self.oclIsTypeOf(TM_Period) /* Placeholder NULL Constraint */
```

- Class in which the example constraint is defined: TM_Period
- Constraint description: The temporal position of the beginning of the period must be less than (i.e., earlier than) the temporal position of the end of the period.
- Additional note on the constraint: ISO 19108 expresses this constraint loosely as { self.begin.position < self.end.position}. The comparison operator is, however, not directly applicable to values of the complex type TM_Position; instead, a set of discriminated comparisons are required between values of its various alternate datatypes: Date, Time, DateTime, and TM_TemporalPosition.

B.2.6.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is not possible.

The same explanation as for Property Co-constraint (related entity, numeric comparison) applies.

B.2.7. Related Datatype Use Required (at least one)

B.2.7.1. OCL constraint

```
inv: HydroVertDimIntervalMeta.allInstances().soundingMetadata->exists(sm|sm=self) or
HydroVertDimMeta.allInstances().soundingMetadata->exists(sm|sm=self)
```

- Class in which the example constraint is defined: SoundingMetadata
- Constraint description: There exists at least one of the following associated datatypes: Hydrographic Vertical Dimension Interval with Metadata, or Hydrographic Vertical Dimension or Reason; with Metadata.

B.2.7.2. Translation to OWL

A translation of the OCL constraint to an OWL class expression is not possible, because the operator allInstances() cannot be translated. For further details, see Table 2.

Annex C: Revision History

Table 5. Revision History

Date	Editor	Release	Primary clauses modified	Descriptions
Sep 28, 2018	J. Echterhoff	1.0	all	Preliminary DER for testbed internal review
Oct 25, 2018	J. Echterhoff	1.0	all	Incorporate feedback from testbed internal review, split document into multiple ERs
Nov 15, 2018	J. Echterhoff	1.0	1.2, 5.2, 6	Incorporate feedback from review by Geosemantics DWG
Dec 04, 2018	J. Echterhoff	1.0	throughout	Incorporate feedback from review by OGC IP team

Annex D: Bibliography

1. Fu, C., Yang, D., Zhang, X., Hu, H.: An approach to translating OCL invariants into OWL 2 DL axioms for checking inconsistency. Automated Software Engineering. 24, 295–339 (2017).
2. Atkinson, C., Kiko, K.: A detailed comparison of UML and OWL. (2005).