Publications Office of the European Union

Initiative for developing eProcurement Ontology

Transformation of the eProcurement UML model into a formal OWL ontology

Disclaimer

This report was prepared for the Publications Office of the European Union by Infeurope.

Document metadata

Reference	WP 1.6: Transformation of the eProcurement UML model into a formal OWL ontology
Corporate Author	Publications Office of the European Union
Author	Eugeniu Costetchi
Reviewers	Natalie Muric, Ioannis Rousochatzakis, George Vernardos
Contractor	Infeurope S.A.
Framework contract	10688/35368
Work package	WP 1.6
Delivery date	20 May 2020
Suggested readers	project partners, future users, legal practitioners, software developers and architects

Abstract

Publications Office of the European Union set off to build an eProcurement ontology. The ultimate objective of the project is to put forth a commonly agreed ontology that will conceptualise, formally encode and make available in an open, structured and machine-readable format data about public procurement, covering end-to-end procurement, i.e. from notification, through tendering to awarding, ordering, invoicing and payment.

The process and the methodology adopted involve modelling the conceptual model in Unified Modelling Language (UML) and then, by abiding a set of conventions and recommendations, transform that model into a formal ontology expressed in Web Ontology Language (OWL).

This document provides a working definition of the transformation rules from the UML conceptual model into the formal OWL ontology and validation data shapes. These rules are organised in accordance with the eProcurement ontology architecture.

Contents

1 Introduction

Publications Office of the European Union set off to build an eProcurement ontology well motivated in [6, p.5-9]. The ultimate objective of the project is to put forth a commonly agreed ontology that will conceptualise, formally encode and make available in an open, structured and machine-readable format data about public procurement, covering end-to-end procurement, i.e. from notification, through tendering to awarding, ordering, invoicing and payment [12].

The process and the methodology adopted involve modelling the conceptual model in Unified Modelling Language (UML) [5] and then, by abiding a set of conventions and recommendations, transform that model into a formal ontology [6, p.12-21] expressed in Web Ontology Language (OWL) [15].

This document provides a working definition of the transformation rules from the UML conceptual model into the formal ontology and validation data shapes. These rules are organised in accordance with the eProcurement ontology architecture [6, p.21-27].

1.1 State of the art

Much has been written about correspondences and between and transformation from UML to OWL and vice versa [16]. The most significant literature on this topic was published between 2006 and 2019 comprising three book chapters, nine journal papers and multiple conference papers.

The work presented in [10] transforms into OWL some selected elements of UML models containing multiple UML class, object and state-chart diagrams in order to analyse consistency of the models. A similar approach is presented in [11], which is focused on detecting inconsistency in models containing UML class and state-chart diagrams.

The papers [8, 21, 22] investigate the differences and similarities between UML and OWL in order to present transformations of selected (and identified as useful) elements of UML class diagram. In [22], the need for UML–OWL transformation is additionally motivated by not repeating the modelling independently in both languages.

The paper [1] compares OWL abstract syntax elements to the equivalent UML features and appropriate OCL statements. The analysis is conducted in the direction from OWL to UML. For every OWL construct its UML interpretation is proposed.

The works presented in [20, 19, 14] are focused on extracting ontological knowledge from UML class diagrams and describe some UML–OWL mappings with the aim to reuse the existing UML models and stream the building of OWL domain ontologies.

In [16] is presented a comprehensive review of the related work. We use it as a guideline for ensuring the necessary coverage of the transformation rules specified in this report. It is important to note here that all the UML elements are treated here, but only the ones employed in the eProcurement conceptual model.

1.2 How to read this document

The reminder of this document comprises four sections covering major UML aspects. Section 2 treats classes and attributes. Following section deals with the main connector types employed in the eProcurement model, namely: associations, dependencies and generalisations. Section 4 explains how the datatypes and enumerations should be transformed and, finally, Section 5 provides a few transformation rules that are applicable to all UML elements and are concerned with comments, labels and notes.

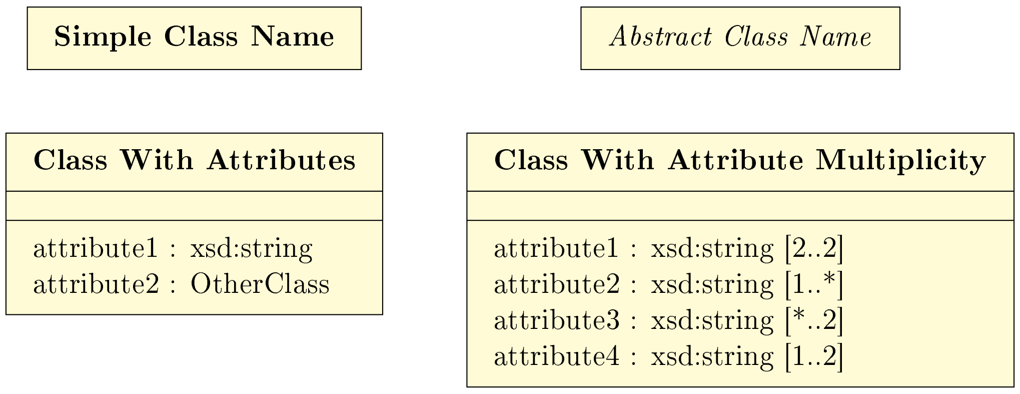
Each section provides a table with overview of the transformation rule set comprised within. The table provides three columns, one for every layer of the ontology architecture comprising rules: (a) in core ontology layer, (b) in data shape layer, and (c) in reasnoning layer.

Transformation rules are specified in a normative language and are aided by prototypical UML diagram fragments (usually preceding the rule) along with representation of the corresponding OWL fragment depicted in Graffoo visual notation [9]. The diagrams are provided side to side in order to increase comprehension, with UML fragment on the left constituting the source of the transformation and the OWL fragment on the right representing the final result of the transformation.

Each transformation rule is accompanied by the formal OWL representation, in Turtle [4] and RDF/XML [17, 2] syntaxes, corresponding to the depicted UML fragment from the preceding figure. RDF/XML is a syntax to express RDF graphs as an XML document. Turtle is a textual syntax for RDF [18] that is compact and resembles a natural text form with abbreviations for common usage patterns and datatypes.

1.3 UML visual notation

This section provides the main UML elements employed in this document. A detailed description can be consulted in the standard specifications [5] and in the user guide [3].



Simple Class Name
Abstract Class Name
Class With Attributes
attribute1 : xsd:string
attribute2 : OtherClass
Class With Attribute Multiplicity
attribute1 : xsd:string [2..2]
attribute2 : xsd:string [1..*]
attribute3 : xsd:string [*..2]
attribute4 : xsd:string [1..2]

Figure 1: UML visual notation for classes and attributes

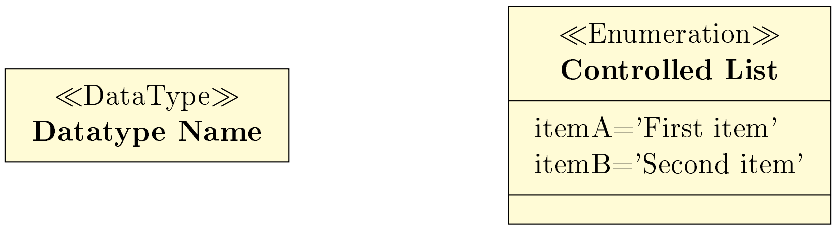


Figure 2: UML visual notation for datatypes and enumerations

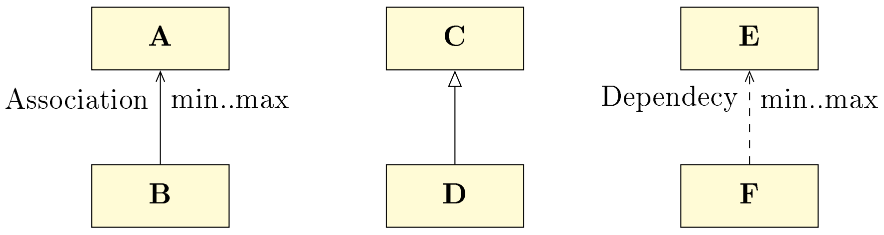


Figure 3: UML visual notation for association, generalisation, and dependency

Figure 1 depicts simple, abstract and regular classes with and without attribute specifications. Note that no class methods are ever employed as this document as the transformations aim at data structures only.

Figure 2 depicts a primitive datatype and an enumeration. No complex datatypes are depicted as they are treated in the same manner as classes are.

Figure 3 depicts association, generalisation and dependency connectors as the only ones necessary to model the eProcurement conceptual model.

1.4 Graffoo visual notation

This section provides the main Graffoo elements employed in this document. A detailed description can be consulted in the OWL standard specifications [15] and in the Graffoo user guide [9].

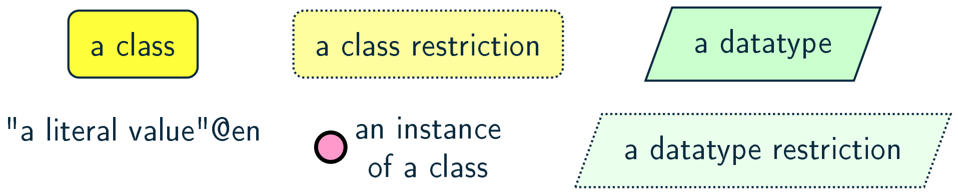


Figure 4: Graffoo visual notation for classes, instances and datatypes

A yellow rectangle with solid black border is used to declare classes. Solid black and labelled arrows are used to declare class axioms. A green rhomboid with solid

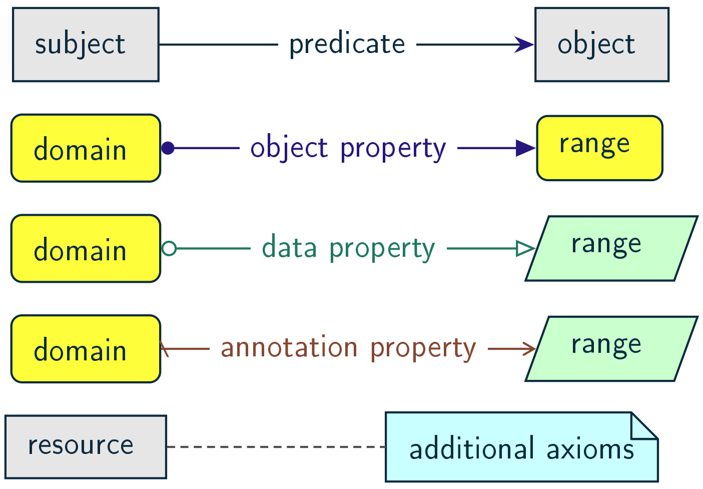


Figure 5: Graffoo visual notation for object and data properties and generic links

black border is used to declare datatypes. Solid black and labelled arrows are used to declare class axioms.

A pink circle with solid black border is used to declare individuals. Solid black and labelled arrows are used to declare axioms and assertions among individuals.

A green solid line is used to declare data properties, where the empty circle at the beginning identifies the property domain while the empty arrow at the end indicates the property range. A blue solid line is used to declare object properties, where the solid circle at the beginning identifies the property domain while the solid arrow at the end indicates the property range.

The following sections present the transformation rules necessary for converting the UML eProcurement conceptual model into a formal OWL ontology.

2 Transformation of UML classes and attributes

In this section are specified transformation rules for UML class and attribute elements. Table 1 provides an overview of the section coverage.

UML element	Rules in core ontology layer	Rules in data shape layer	Rules in reasoning layer
Class	Rule 1	Rule 2	
Abstract class		Rule 3	
Attribute	Rule 4		Rule 5
Attribute type		Rule 7	Rule 6
Attribute multiplicity		Rule 8	Rule 9, 10

Table 1: Overview of transformation rules for UML classes and attributes

2.1 Class

In UML, a Class [5] is purposed to specify a classification of objects. UML represents atomic classes as named elements of type *Class* without further features. In OWL, the atomic class, `owl:Class`, has no intension. It can only be interpreted by its name that has a meaning in the world outside the ontology. The atomic class is a class description that is simultaneously a class axiom [1].

ClassName
:ClassName

Figure 6: Visual representation of a class in UML (on the left) and OWL (on the right)

Rule 1 (Class – in core ontology layer)**.** Specify declaration axiom for UML Class as OWL Class where the URI and a label are deterministically generated from the class name. The label and, if available, the description are ascribed to the class.

```
:ClassName a owl:Class ;
  rdfs:label "Class name"@en ;
.
```

Listing 1: Class declaration in Turtle syntax

```
<owl:Class
    rdf:about="http://base.uri/ClassName">
  <rdfs:label xml:lang="en">Class
      name</rdfs:label>
</owl:Class>
```

Listing 2: Class declaration in RDF/XML syntax

Rule 2 (Class – in data shape layer)**.** Specify declaration axiom for UML Class as SHACL Node Shape where the URI and a label are deterministically generated from the class name.

```
:ClassName a sh:NodeShape .
```

Listing 3: Node shape declaration in Turtle syntax

```
<rdf:Description
    rdf:about="http://base.uri/ClassName">
<rdf:type
    rdf:resource="http://www.w3.org/ns/shacl#NodeShape">
<rdf:Description>
```

Listing 4: Node shape declaration in RDF/XML syntax

2.2 Abstract class

In UML, an abstract Class [5] cannot have any instances and only its subclasses can be instantiated. The abstract classes are declared just like the regular ones (Rule 1 and 2) and in addition a constraint validation rule is generated to ensure that no instance of this class is permitted.

OWL follows the Open World Assumption [15], therefore, even if the ontology does not contain any instances for a specific class, it is unknown whether the class has any instances. We cannot confirm that the UML abstract class is correctly defined with respect to the OWL domain ontology, but we can detect if it is not using SHACL constraints.

ClassName
:ClassName

Figure 7: Visual representation of an abstract class in UML (on the left) and OWL (on the right)

Rule 3 (Class – in data shape layer)**.** Specify declaration axiom for UML Class as SHACL Node Shape with a SPARQL constraint that selects all instances of this class.

```
:ClassName
  rdf:type sh:NodeShape ;
  sh:sparql [
    sh:select """SELECT $this
      WHERE {
        $this a :ClassName .
      }
      """ ;
   ] ;
.
```

Listing 5: Instance checking constraint in Turtle syntax

```
<sh:NodeShape
    rdf:about="http://base.uri/ClassName">
  <sh:sparql rdf:parseType="Resource">
    <sh:select>SELECT $this
      WHERE {
      $this a :ClassName .
      }
    </sh:select>
  </sh:sparql>
</sh:NodeShape>
```

Listing 6: Instance checking constraint in RDF/XML syntax

2.3 Attribute

The UML attributes [5] are properties that are owned by a Classifier, e.g. Class. Both UML attributes and associations are represented by one meta-model element – Property. OWL also allows one to define properties. A transformation of UML attribute to OWL data property or OWL object property bases on its type. If the type of the attribute is a primitive type it should be transformed into OWL datatype property. However, if the type of the attribute is a structured datatype, class of enumeration , it should be transformed into an OWL object property.

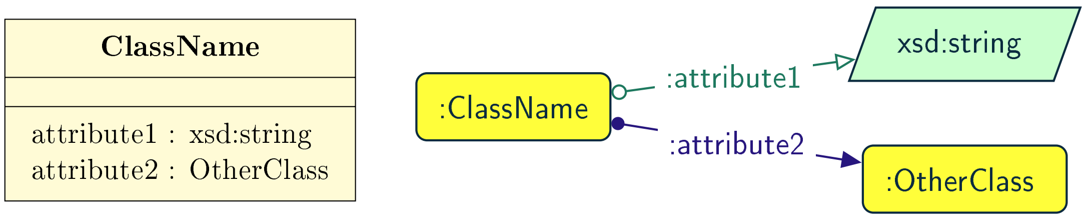


Figure 8: Visual representation of class attributes in UML (on the left) and OWL properties (on the right)

Rule 4 (Attribute – in core ontology layer)**.** Specify declaration axiom(s) for attribute(s) as OWL data or object properties deciding based on their types. The attributes with primary types should be treated as data properties, whereas those typed with classes or enumerations should be treated as object properties.

```
:attribute1 a owl:DatatypeProperty ;
  rdfs:label "attribute 1"@en;
  skos:definition "Description of the
      attribute meaning"@en;
.
:attribute2 a owl:ObjectProperty ;
  rdfs:label "attribute 2"@en;
  skos:definition "Description of the
      attribute meaning"@en;
.
```

Listing 7: Property declaration in Turtle syntax

```
<owl:DatatypeProperty
    rdf:about="http://base.uri/attribute1">
  <rdfs:label xml:lang="en">attribute
      1</rdfs:label>
  <skos:definition
      xml:lang="en">Description of the
      attribute meaning</skos:definition>
</owl:DatatypeProperty>
<owl:ObjectProperty
    rdf:about="http://base.uri/attribute2">
  <rdfs:label xml:lang="en">attribute
      1</rdfs:label>
  <skos:definition
      xml:lang="en">Description of the
      attribute meaning</skos:definition>
</owl:ObjectProperty>
```

Listing 8: Property declaration in RDF/XML syntax

2.4 Attribute owner

Rule 5 (Attribute domain – in reasnoning layer)**.** Specify data (or object) property domains for attribute(s).

```
:attribute1 a
    owl:DatatypeProperty ;
  rdfs:domain :ClassName ;
.
:attribute2 a owl:ObjectProperty ;
  rdfs:domain :ClassName ;
.
```

Listing 9: Domain specification in Turtle syntax

```
<owl:DatatypeProperty
    rdf:about="http://base.uri/attribute1">
  <rdfs:domain
      rdf:resource="http://base.uri/ClassName"/>
</owl:DatatypeProperty>
<owl:ObjectProperty
    rdf:about="http://base.uri/attribute2">
  <rdfs:domain
      rdf:resource="http://base.uri/ClassName"/>
</owl:ObjectProperty>
```

Listing 10: Domain specification in RDF/XML syntax

2.5 Attribute type

Rule 6 (Attribute type – in reasnoning layer)**.** Specify data (or object) property range for attribute(s).

```
:attribute1 a
    owl:DatatypeProperty;
  rdfs:range xsd:string;
.
:attribute2 a owl:ObjectProperty;
  rdfs:range :OtehrClass;
.
```

Listing 11: Range specification in Turtle syntax

```
<owl:DatatypeProperty
    rdf:about="http://base.uri/attribute1">
  <rdfs:range
      rdf:resource="http://www.w3c.org...#string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty
    rdf:about="http://base.uri/attribute2">
  <rdfs:range
      rdf:resource="http://base.uri/OtherClass"/>
</owl:ObjectProperty>
```

Listing 12: Range specification in RDF/XML syntax

Rule 7 (Attribute range shape – in data shape layer)**.** Within the SHACL Node Shape corresponding to the UML class, specify property constraints, for each UML attribute, indicating the range class or datatype.

```
:ClassName a sh:NodeShape ;
  sh:property [
    a sh:PropertyShape ;
    sh:path :attribute1 ;
    sh:datatype xsd:string ;
    sh:name "attribute 1" ;
  ];
  sh:property [
    a sh:PropertyShape ;
    sh:path :attribute2 ;
    sh:class :OtherClass ;
    sh:name "attribute 2" ;
  ];
.
```

Listing 13: Property class and datatype constraint in Turtle syntax

```
<sh:NodeShape
    rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/attribute1"/>
    <sh:name>attribute 1</sh:name>
    <sh:datatype
        rdf:resource="http://www.w3c.org...#string"/>
  </sh:PropertyShape>
</sh:property>
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/attribute2"/>
    <sh:name>attribute 2</sh:name>
    <sh:class
        rdf:resource="http://base.uri/OtherClass"/>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 14: Property class and datatype constraint in RDF/XML syntax

2.6 Attribute multiplicity

In [5], multiplicity bounds of multiplicity element are specified in the form of `[<lower-bound> .. <upper-bound>]`. The lower-bound, also referred here as minimum cardinality or `min` is of a non-negative Integer type and the upper-bound, also referred here as maximum cardinality or `max`, is of an UnlimitedNatural type (see Section 4.1). The strictly compliant specification of UML in version 2.5 defines only a single value range for MultiplicityElement. not limit oneself to a single interval. Therefore, the below UML to OWL mapping covers a wider case – a possibility of specifying more value ranges for a multiplicity element. Nevertheless, if the reader would like to strictly follow the current UML specification, the particular single lower..upper bound interval is therein also comprised.

Rule 8 (Attribute multiplicity – in data shape layer)**.** Within the SHACL Node Shape corresponding to the UML class, specify property constraints, corresponding to each attribute, indicating the minimum and maximum cardinality, only where min

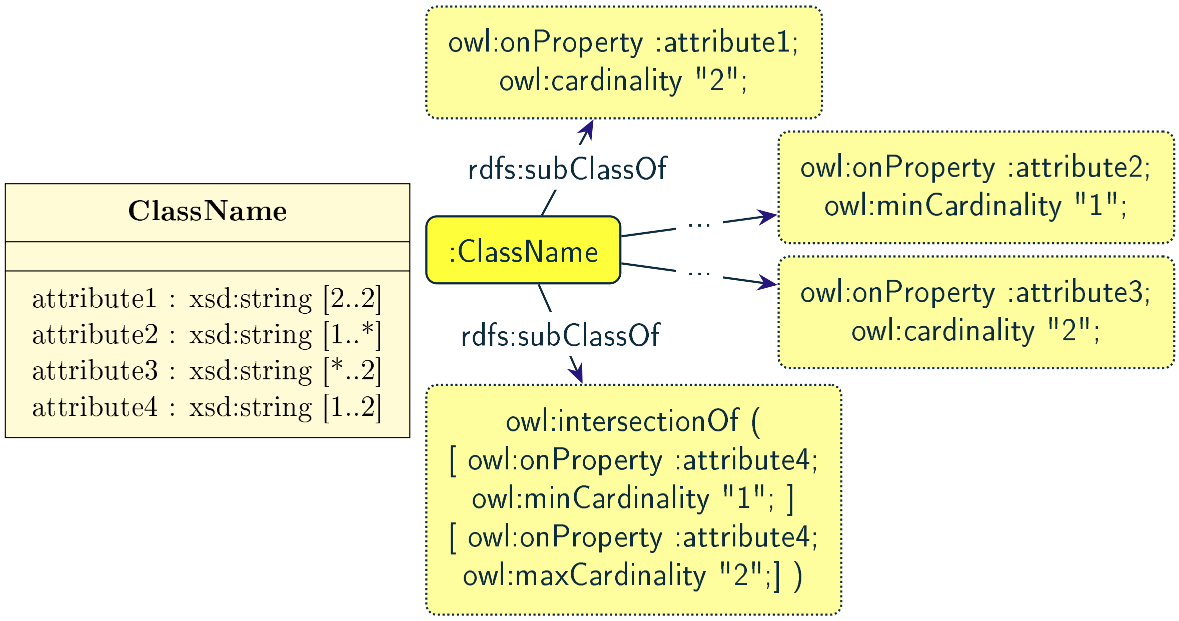

ClassName
attribute1 : xsd:string [2..2]
attribute2 : xsd:string [1..*]
attribute3 : xsd:string [*..2]
attribute4 : xsd:string [1..2]
:ClassName
rdfs:subClassOf
owl:onProperty :attribute1; owl:cardinality "2";
owl:onProperty :attribute2; owl:minCardinality "1";
owl:onProperty :attribute3; owl:cardinality "2";
rdfs:subClassOf
owl:intersectionOf ([owl:onProperty :attribute4; owl:minCardinality "1";] [owl:onProperty :attribute4; owl:maxCardinality "2";])

Figure 9: Visual representation of class attributes with multiplicity in UML (on the left) and OWL class specialising an anonymous restriction of properties (on the right)

and max are different from "*" (any) and multiplicity is not [1..1]. The expressions are formulated according to the following cases.

A. exact cardinality, e.g. [2..2]

B. minimum cardinality only, e.g. [1..*]

C. maximum cardinality only, e.g. [*..2]

D. minimum and maximum cardinality , e.g. [1..2]

```
:ClassName a sh:NodeShape ;
  sh:property [
      sh:path :attribute1;
      sh:minCount 2 ;
      sh:maxCount 2 ;
      sh:name "attribute 1" ;
    ] ;
.
```

Listing 15: Exact cardinality constraint in Turtle syntax

```
<sh:NodeShape rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/attribute1"/>
    <sh:name>attribute 1</sh:name>
    <sh:minCount
        rdf:datatype="http://www.w3.org...#integer"
        >2</sh:minCount>
    <sh:maxCount
        rdf:datatype="http://www.w3.org...#integer"
        >2</sh:maxCount>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 16: Exact cardinality constraint in RDF/XML syntax

```
:ClassName a sh:NodeShape ;
  sh:property [
      sh:path :attribute2;
      sh:minCount 1 ;
      sh:name "attribute 2" ;
    ] ;
.
```

Listing 17: Min cardinality constraint in Turtle syntax

```
<sh:NodeShape rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/attribute2"/>
    <sh:name>attribute 2</sh:name>
    <sh:minCount
        rdf:datatype="http://www.w3.org...#integer"
        >1</sh:minCount>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 18: Min cardinality constraint in RDF/XML syntax

```
:ClassName a sh:NodeShape ;
  sh:property [
      sh:path :attribute3;
      sh:maxCount 2 ;
      sh:name "attribute 3" ;
    ] ;
.
```

Listing 19: Max cardinality constraint in Turtle syntax

```
<sh:NodeShape rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/attribute3"/>
    <sh:name>attribute 3</sh:name>
    <sh:maxCount
        rdf:datatype="http://www.w3.org...#integer"
        >2</sh:maxCount>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 20: Max cardinality constraint in RDF/XML syntax

```
:ClassName a sh:NodeShape ;
  sh:property [
      sh:path :attribute4;
      sh:minCount 1 ;
      sh:maxCount 2 ;
      sh:name "attribute 4" ;
    ] ;
.
```

Listing 21: Min and max cardinality constraint in Turtle syntax

```
<sh:NodeShape rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/attribute4"/>
    <sh:name>attribute 4</sh:name>
    <sh:minCount
        rdf:datatype="http://www.w3.org...#integer"
        >1</sh:minCount>
    <sh:maxCount
        rdf:datatype="http://www.w3.org...#integer"
        >2</sh:maxCount>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 22: Min and max cardinality constraint in RDF/XML syntax

It should be noted that upper-bound of UML MultiplicityElement can be specified as unlimited: “*”. In OWL, cardinality expressions serve to restrict the number of individuals that are connected by an object property expression to a given number of instances of a specified class expression [15]. Therefore, UML unlimited upper-bound does not add any information to OWL ontology, hence it is not transformed.

Rule 9 (Attribute multiplicity – in reasnoning layer)**.** For each attribute multiplicity of the form (min .. max), where min and max are different than “*” (any), specify a subclass axiom where the OWL class, corresponding to the UML class, specialises an anonymous restriction of properties formulated according to the following cases.

A. exact cardinality, e.g. [2..2]

B. minimum cardinality only, e.g. [1..*]

C. maximum cardinality only, e.g. [*..2]

D. maximum and maximum cardinality , e.g. [1..2]

```
:ClassName a owl:Class ;
  rdfs:subClassOf [ a
      owl:Restriction ;
      owl:cardinality
          "2"^^xsd:integer;
      owl:onProperty :attribute1 ;
    ] ;
.
```

Listing 23: Cardinality restriction in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
    rdf:resource="http://base.uri/attribute1"/>
      <owl:cardinality
          rdf:datatype="http://www.w3.org...#integer"
          >2</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 24: Cardinality restriction in RDF/XML syntax

```
:ClassName a owl:Class ;
  rdfs:subClassOf [ a
      owl:Restriction ;
      owl:minCardinality
          "1"^^xsd:integer;
      owl:onProperty :attribute2 ;
    ] ;
.
```

Listing 25: Min cardinality restriction in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
    rdf:resource="http://base.uri/attribute2"/>
      <owl:minCardinality
          rdf:datatype="http://www.w3.org...#integer"
          >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 26: Min cardinality restriction in RDF/XML syntax

```
:ClassName a owl:Class ;
  rdfs:subClassOf [ a
      owl:Restriction ;
      owl:maxCardinality
          "2"^^xsd:integer;
      owl:onProperty :attribute3 ;
    ] ;
.
```

Listing 27: Max cardinality restriction in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
    rdf:resource="http://base.uri/attribute3"/>
      <owl:maxCardinality
          rdf:datatype="http://www.w3.org...#integer"
          >2</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 28: Max cardinality restriction in RDF/XML syntax

```
:ClassName a owl:Class ;
  rdfs:subClassOf [
    rdf:type owl:Class ;
    owl:intersectionOf (
      [ a owl:Restriction ;
        owl:minCardinality
            "1"^^xsd:integer;
        owl:onProperty
            :attribute4; ]
      [ a owl:Restriction ;
        owl:maxCardinality
            "2"^^xsd:integer;
        owl:onProperty
            :attribute4; ]
      ) ;
    ] ;
.
```

Listing 29: Min and max cardinality restriction in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty
    rdf:resource="http://base.uri/attribute4"/>
          <owl:minCardinality
              rdf:datatype="...#integer"
          >1</owl:minCardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty
    rdf:resource="http://base.uri/attribute4"/>
          <owl:maxCardinality
              rdf:datatype="...#integer"
          >2</owl:maxCardinality>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 30: Min and max cardinality restriction in RDF/XML syntax

Attributes with multiplicity exactly one correspond to functional object or data properties in OWL. If we apply the previous rule specifying min and max cardinality will lead to inconsistent ontology. To avoid that it is important that min and max cardinality are not generated from [1..1] multiplicity but only functional property axiom.

Rule 10 (Attribute multiplicity "one" – in reasnoning layer)**.** For each attribute that has multiplicity exactly one, i.e. [1..1], specify functional property axiom.

```
:attribute5 a
    owl:FunctionalProperty .
```

Listing 31: Declaring a functional property in Turtle syntax

```
<owl:FunctionalProperty
    rdf:about="http://base.uri/attribute5"/>
```

Listing 32: Declaring a functional property in RDF/XML syntax

3 Transformation of UML connectors

In this section are specified transformation rules for UML association, generalisation and dependency connectors. Table 2 provides an overview of the section coverage.

UML element	Rules in core ontology layer	Rules in data shape layer	Rules in reasoning layer
Association*	Rule 11		
Association domain*			Rule 12
Association range*		Rule 14	Rule 13
Association multiplicity*		Rule 15	Rule 16, 17
Association asymmetry*		Rule 18	Rule 19
Association inverse†			Rule 20
Dependency‡	...	...	...
Class generalisation	Rule 21		
Property generalisation	Rule 22		
Class equivalence			Rule 23
Property equivalence			Rule 24

Table 2: Transformation rules overview for UML connectors

*Applicable to unidirectional and bidirectional connectors
†Applicable to bidirectional connectors only
‡Inherits all the rules from unidirectional and bidirectional associations

3.1 Unidirectional association

A binary Association specifies a semantic relationship between two member ends represented by properties. Please note that in accordance with specification [5], the association end names are not obligatory. However, we adhere to the UML conventions [7], where specification of at one member ends, for unidirectional association, and two member ends, for bidirectional association, is mandatory. Moreover, provision of a connector (general) name is discouraged.

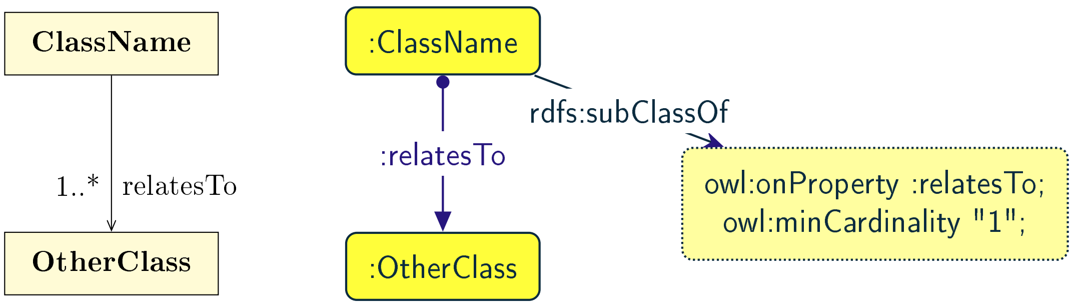


Figure 10: Visual representation of an UML unidirectional association (on the left) and an OWL property with cardinality restriction on domain class (on the right)

Rule 11 (Unidirectional association – in core ontology layer). Specify object property declaration axiom for the target end of the association.

```
:relatesTo a owl:DatatypeProperty
    ;
  rdfs:label "relates oo"@en;
  skos:definition "Description of
      the relationship
      meaning"@en;
.
```

Listing 33: Property declaration in Turtle syntax

```
<owl:DatatypeProperty
    rdf:about="http://base.uri/relatesTo">
  <rdfs:label xml:lang="en">relates to</rdfs:label>
  <skos:definition xml:lang="en">Description of the
      relationship meaning</skos:definition>
</owl:DatatypeProperty>
```

Listing 34: Property declaration in RDF/XML syntax

3.2 Association source

Rule 12 (Association source – in reasnoning layer). Specify object property domain for the target end of the association.

```
:relatesTo a owl:ObjectProperty ;
  rdfs:domain :ClassName ;
.
```

Listing 35: Domain specification in Turtle syntax

```
<owl:ObjectProperty
    rdf:about="http://base.uri/relatesTo">
  <rdfs:domain
      rdf:resource="http://base.uri/ClassName"/>
</owl:ObjectProperty>
```

Listing 36: Domain specification in RDF/XML syntax

3.3 Association target

Rule 13 (Association target – in reasnoning layer). Specify object property range for the target end of the association.

```
:relatesTo a owl:ObjectProperty ;
  rdfs:range :ClassName ;
.
```

Listing 37: Range specification in Turtle syntax

```
<owl:ObjectProperty
    rdf:about="http://base.uri/relatesTo">
  <rdfs:range
      rdf:resource="http://base.uri/ClassName"/>
</owl:ObjectProperty>
```

Listing 38: Range specification in RDF/XML syntax

Rule 14 (Association range shape – in data shape layer). Within the SHACL Node Shape corresponding to the source UML class, specify property constraints indicating the range class.

```
:ClassName a sh:NodeShape ;
  sh:property [
    a sh:PropertyShape ;
    sh:path :relatesTo ;
    sh:class :OtherClass ;
    sh:name "relates to" ;
  ];
.
```

Listing 39: Property class constraint in Turtle syntax

```
<sh:NodeShape rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/relatesTo"/>
    <sh:name>relates to</sh:name>
    <sh:class
        rdf:resource="http://base.uri/OtherClass"/>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 40: Property class constraint in RDF/XML syntax

3.4 Association multiplicity

Rule 15 (Association multiplicity – in data shape layer). Within the SHACL Node Shape corresponding to the source UML class, specify property constraints indicating minimum and maximum cardinality according to cases provided by Rule 8.

```
:ClassName a sh:NodeShape ;
  sh:property [
      sh:path :relatesTo;
      sh:minCount 1 ;
      sh:name "relates to" ;
    ] ;
.
```

Listing 41: Min cardinality constraint in Turtle syntax

```
<sh:NodeShape rdf:about="http://base.uri/ClassName">
<sh:property>
  <sh:PropertyShape>
    <sh:path
    rdf:resource="http://base.uri/relatesTo"/>
    <sh:name>relates to</sh:name>
    <sh:minCount
        rdf:datatype="http://www.w3.org...#integer"
        >1</sh:minCount>
  </sh:PropertyShape>
</sh:property>
</sh:NodeShape>
```

Listing 42: Min cardinality constraint in RDF/XML syntax

Rule 16 (Association multiplicity – in reasnoning layer). For the association target multiplicity, where min and max are different than "*" (any) and multiplicity is not [1..1], specify a subclass axiom where the source class specialises an anonymous restriction of properties formulated according to cases provided by Rule 9.

```
:ClassName a owl:Class ;
  rdfs:subClassOf [ a
      owl:Restriction ;
      owl:minCardinality
          "1"^^xsd:integer;
      owl:onProperty :relatesTo ;
    ] ;
.
```

Listing 43: Min cardinality restriction in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
    rdf:resource="http://base.uri/relatesTo"/>
      <owl:minCardinality
          rdf:datatype="http://www.w3.org...#integer"
          >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Listing 44: Min cardinality restriction in RDF/XML syntax

Rule 17 (Association multiplicity "one" – in reasnoning layer). If the association multiplicity is exactly one, i.e. [1..1], specify functional property axiom like in Rule 10.

```
:relatesTo a owl:FunctionalProperty .
```

Listing 45: Declaring a functional property in Turtle syntax

```
<owl:FunctionalProperty
    rdf:about="http://base.uri/relatesTo"/>
```

Listing 46: Declaring a functional property in RDF/XML syntax

3.5 Recursive association

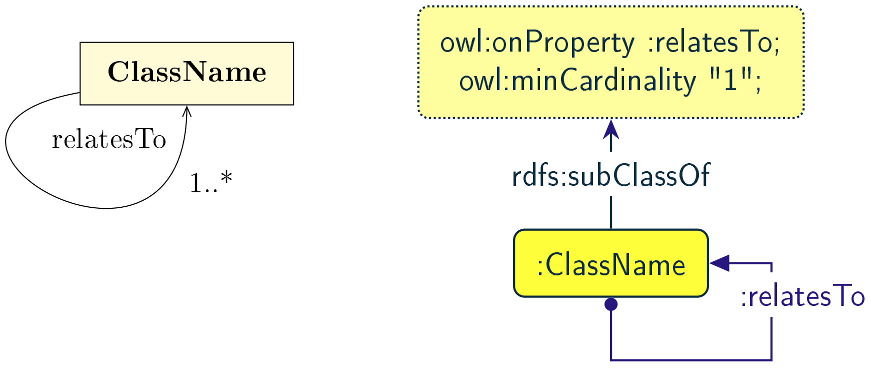


Figure 11: Visual representation of an UML recursive association (on the left) and OWL recursive properties with cardinality restrictions on domain class (on the right)

In case of recursive associations, that are from one class to itself, the transformation rules must be applied as in the case of regular unidirectional association, which are from Rule 11 to Rule 17. In addition the association must be marked as asymmetric expressed in Rule 19.

Rule 18 (Association asymmetry – in data shape layer). Within the SHACL Node Shape corresponding to the UML class, specify SPARQL constraint selecting instances connected by the object property in a reciprocal manner.

```
:ClassName a sh:NodeShape ;
  sh:sparql [
    sh:select """
       SELECT ?this ?that
       WHERE {
       ?this :relatesTo ?that .
       ?that :relatesTo this .
       }""" ; ] ;
.
```

Listing 47: Declaring an asymmetric property in Turtle syntax

```
<sh:NodeShape
    rdf:about="http://base.uri/ClassName">
  <sh:sparql rdf:parseType="Resource">
    <sh:select>
      SELECT ?this ?that
      WHERE {
      ?this :relatesTo ?that .
      ?that :relatesTo ?this .}
    </sh:select>
  </sh:sparql>
</sh:NodeShape>
```

Listing 48: Declaring an asymmetric property in RDF/XML syntax

Rule 19 (Association asymmetry – in reasnoning layer). Specify the asymmetry object property axiom for each end of a recursive association.

```
:relatesTo a owl:AsymmetricProperty .
```

Listing 49: Declaring an asymmetric property in Turtle syntax

```
<owl:AsymmetricProperty
    rdf:about="http://base.uri/relatesTo"/>
```

Listing 50: Declaring an asymmetric property in RDF/XML syntax

3.6 Bidirectional association

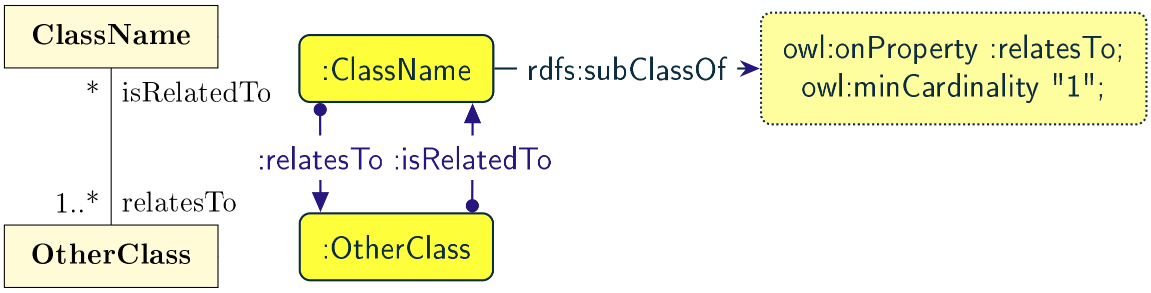


Figure 12: Visual representation of an UML bidirectional association (on the left) and OWL properties with cardinality restrictions on domain class (on the right)

The bidirectional associations should be treated, both on source and target ends, like two unidirectional associations. The transformation rules from Rule 11 to Rule 17 must be applied to both ends. In addition these rule the inverse relation axiom must be specified.

Rule 20 (Association inverse – in reasnoning layer)**.** Specify inverse object property between the source and target ends of the association.

```
:relatesTo owl:inverseOf
    :isRelatedTo .
```

Listing 51: Declaring an inverse property in Turtle syntax

```
<owl:ObjectProperty
    rdf:about="http://base.uri/relatesTo">
  <owl:inverseOf
      rdf:resource="http://base.uri/isRelatedTo"/>
</owl:ObjectProperty>
```

Listing 52: Declaring an inverse property in RDF/XML syntax

3.7 Unidirectional dependency

The UML dependency connectors should be transformed by the rules specified for UML association connectors.

3.8 Class generalisation

Generalisation [5] defines specialization relationship between Classifiers. In case of UML classes it relates a more specific Class to a more general Class.

UML generalisation set [5] groups generalizations; incomplete and disjoint constraints indicate that the set is not complete and its specific Classes have no common instances. The UML conventions [7] specify that all sibling classes are by default disjoint, therefore even if no generalisation set is provided it is assumed to be implicit.

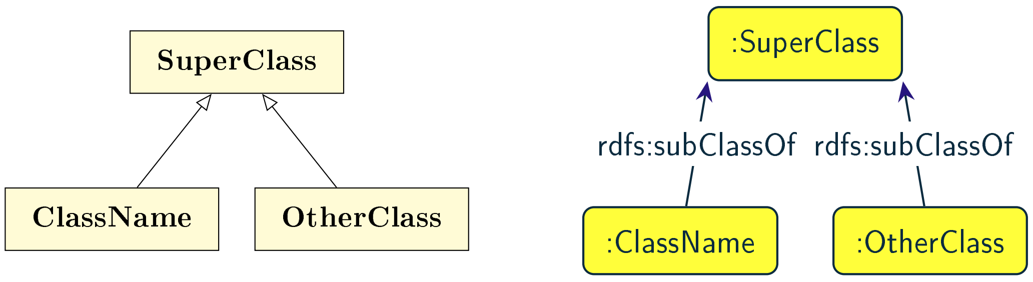


Figure 13: Visual representation of UML generalisation (on the left) and OWL subclass relation (on the right)

Rule 21 (Class generalisation – in core ontology layer)**.** Specify subclass axiom for the generalisation between UML classes. Sibling classes must be disjoint with one another.

```
:ClassName rdfs:subClassOf
    :SuperClass.
:OtherClass rdfs:subClassOf
    :SuperClass;
   owl:disjointWith :ClassName ;
.
```

Listing 53: Sub-classification in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <rdfs:subClassOf
      rdf:resource="http://base.uri/SuperClass"/>
</owl:Class>
<owl:Class
    rdf:about="http://base.uri/OtherClass">
  <rdfs:subClassOf
      rdf:resource="http://base.uri/SuperClass"/>
  <owl:disjointWith
      rdf:resource="http://base.uri/ClassName"/>
</owl:Class>
```

Listing 54: Sub-classification in RDF/XML syntax

3.9 Property generalisation

Generalization [5] defines specialization relationship between Classifiers. In case of the UML associations it relates a more specific Association to more general Association.

Rule 22 (Property generalisation – in core ontology layer)**.** Specify sub-property axiom for the generalisation between UML associations and dependencies.

```
:hasSister rdfs:subPropertyOf
    :relatesTo .
:isSisterOf rdfs:subPropertyOf
    :isRelatedTo .
```

Listing 55: Property specialisation in Turtle syntax

```
<owl:ObjectProperty
    rdf:about="http://base.uri/hasSister">
  <rdfs:subPropertyOf
      rdf:resource="http://base.uri/relatesTo"/>
</owl:ObjectProperty>
<owl:ObjectProperty
    rdf:about="http://base.uri/isSisterOf">
  <rdfs:subPropertyOf
      rdf:resource="http://base.uri/isRelatedTo"/>
</owl:ObjectProperty>
```

Listing 56: Property specialisation in RDF/XML syntax

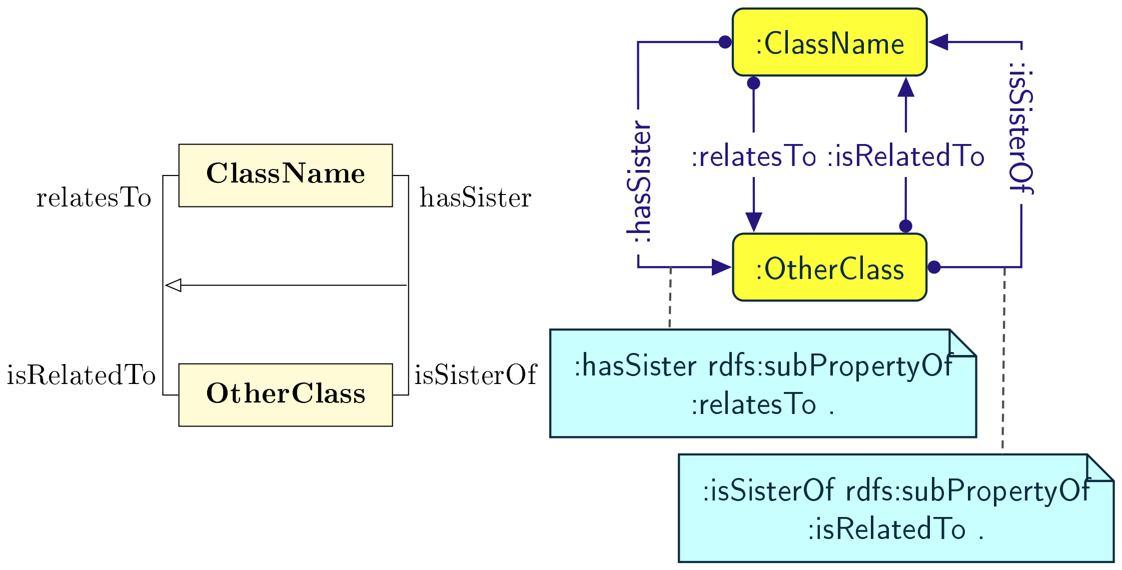


Figure 14: Visual representation of UML property generalisation (on the left) and OWL sub-property relation (on the right)

3.10 Class equivalence

Rule 23 (Equivalent classes – in reasnoning layer)**.** Specify equivalent class axiom for the generalisation with «equivalent» or «complete» stereotype between UML classes.

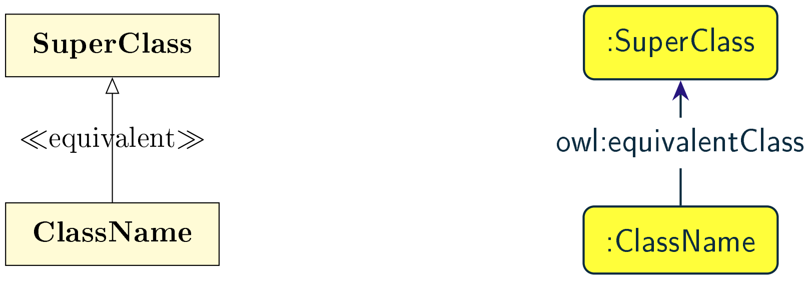


Figure 15: Visual representation of UML class equivalence (on the left) and OWL class equivalence (on the right)

```
:ClassName owl:equivalentClass
    :SuperClass.
```

Listing 57: Class equivalence in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ClassName">
  <owl:equivalentClass
      rdf:resource="http://base.uri/SuperClass"/>
</owl:Class>
```

Listing 58: Class equivalence in RDF/XML syntax

3.11 Property equivalence

Rule 24 (Equivalent properties – in reasnoning layer). Specify equivalent property axiom for the generalisation with «equivalent» or «complete» stereotype between UML properties.

```
:hasSister owl:equivalentProperty
    :relatesTo .
:isSisterOf
    owl:equivalentProperty
    :isRelatedTo .
```

Listing 59: Property equivalence in Turtle syntax

```
<owl:ObjectProperty
    rdf:about="http://base.uri/hasSister">
  <owl:equivalentProperty
      rdf:resource="http://base.uri/relatesTo"/>
</owl:ObjectProperty>
<owl:ObjectProperty
    rdf:about="http://base.uri/isSisterOf">
  <owl:equivalentProperty
      rdf:resource="http://base.uri/isRelatedTo"/>
</owl:ObjectProperty>
```

Listing 60: Property equivalence in RDF/XML syntax

4 Transformation of UML datatypes

In this section are specified transformation rules for UML datatypes and enumerations. Table 3 provides an overview of the section coverage.

UML element	Rules in core ontology layer	Rules in data shape layer	Rules in reasoning layer
Primitive datatype	Rule 25		
Structured datatype	Rule 26		
Enumeration	Rule 27		Rule 29
Enumeration item	Rule 28		

Table 3: Overview of transformation rules for UML datatypes

4.1 Primitive datatype

The UML primitive type defines a predefined datatype without any substructure. The UML specification [5] predefines five primitive types: String, Integer, Boolean, UnlimitedNatural and Real. Here we extended those to the list provided in Table 4.

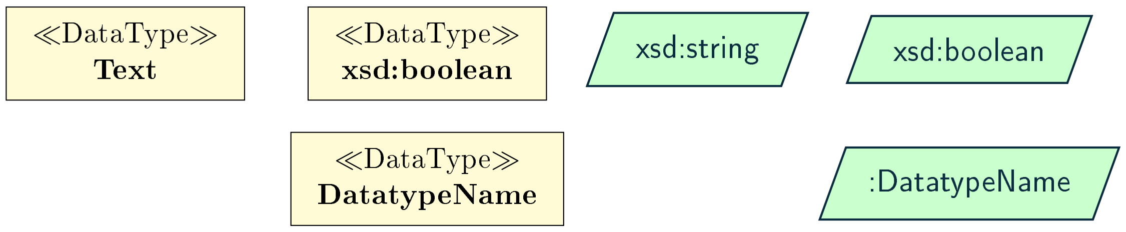


Figure 16: Visual representation of an UML Datatype (on the left) and an OWL Datatype (on the right)

Rule 25 (Datatype – in core ontology layer)**.** Specify datatype declaration axiom for UML datatype as follows:

- UML primitive datatypes are declared as the mapped XSD datatype in Table 4.
- XSD and RDF(S) datatypes are declared as such.
- Model specific datatypes are declared as such.

UML datatype	XSD datatype
Boolean	xsd:boolean
Float	xsd:float
Integer	xsd:integer
Char	xsd:string
String	xsd:string
Short	xsd:short
Long	xsd:long
Decimal	xsd:decimal
Real	xsd:float
Date	xsd:date
Numeric	xsd:integer
Text	xsd:string

Table 4: Mapping of UML primitive types to XSD datatypes

```
xsd:string a rdfs:Datatype ;
  rdfs:label "String"@en ;
  skos:definition "Description of
      the datatype meaning"@en ;
.
xsd:boolean a rdfs:Datatype ;
  rdfs:label "Boolean"@en ;
  skos:definition "Description of
      the datatype meaning"@en ;
.
:DatatypeName a rdfs:Datatype ;
  rdfs:label "Datatype name"@en ;
  skos:definition "Description of
      the datatype meaning"@en ;
.
```

Listing 61: Datatype declaration in Turtle syntax

```
<rdfs:Datatype
    rdf:about="http://www.w3.org/2001/XMLSchema#string">
  <rdfs:label xml:lang="en">String</rdfs:label>
  <skos:definition xml:lang="en">Description of the
      datatype meaning</skos:definition>
</rdfs:Datatype>
<rdfs:Datatype
    rdf:about="http://www.w3.org/2001/XMLSchema#boolean">
  <rdfs:label xml:lang="en">Boolean</rdfs:label>
  <skos:definition xml:lang="en">Description of the
      datatype meaning</skos:definition>
</rdfs:Datatype>
<rdfs:Datatype
    rdf:about="http://base.uri/DatatypeName">
  <rdfs:label xml:lang="en">Datatype name</rdfs:label>
  <skos:definition xml:lang="en">Description of the
      datatype meaning</skos:definition>
</rdfs:Datatype>
```

Listing 62: Datatype declaration in RDF/XML syntax

4.2 Structured datatypes

The UML structured datatype [5] has attributes and is used to define complex data types. The structured datatypes should be treated as classes.

Rule 26 (Structured Datatype – in core ontology layer). Specify OWL class declaration axiom for UML structured datatype.

4.3 Enumeration

UML Enumerations [5] are kinds of datatypes, whose values correspond to one of user-defined literals. They should be transformed into SKOS [13] concept schemes comprising the concepts corresponding to enumerated items.

Rule 27 (Enumeration – in core ontology layer). Specify SKOS concept scheme instantiation axiom for an UML enumeration.

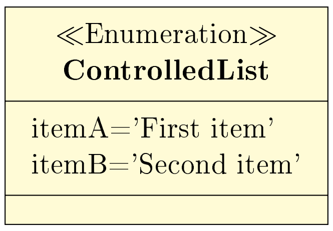


Figure 17: Visual representation of an UML Enumeration

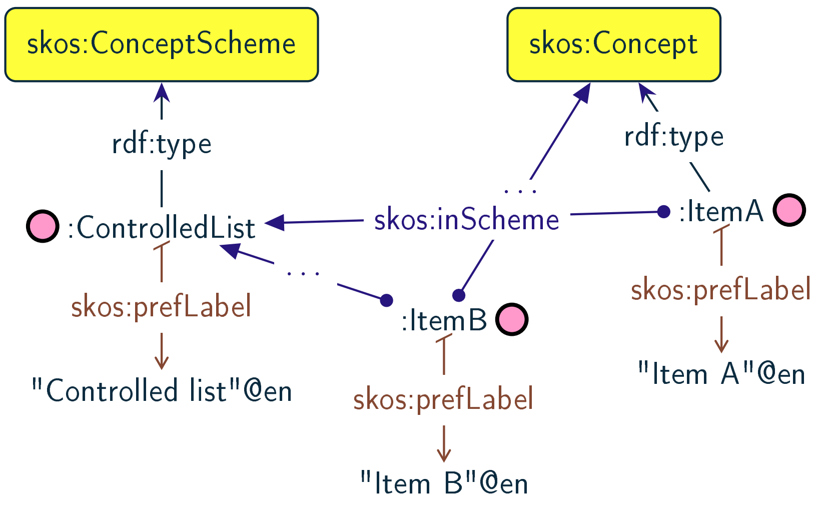


Figure 18: Visual representation of a SKOS concept scheme with concepts

```
:ControlledList a
    skos:ConceptScheme ;
  rdfs:label "Controlled list" ;
  skos:prefLabel "Controlled
      list"@en ;
  skos:definition "Definition of
      the concept scheme
      meaning"@en ;
.
```

Listing 63: Concept scheme instantiation in Turtle syntax

```
<skos:ConceptScheme
    rdf:about="http://base.uri/ControlledList">
  <rdfs:label>Controlled list</rdfs:label>
  <skos:prefLabel xml:lang="en">Controlled
      list</skos:prefLabel>
  <skos:definition xml:lang="en">Definition of the
      concept scheme meaning</skos:definition>
</skos:ConceptScheme>
```

Listing 64: Concept scheme instantiation in RDF/XML syntax

Rule 28 (Enumeration items – in core ontology layer). Specify SKOS concept instantiation axiom for an UML enumeration item.

```
:itemA a skos:Concept ;
  skos:inScheme :ControlledList ;
  rdfs:label "Item A" ;
  skos:prefLabel "Item A"@en ;
  skos:definition "Description fo
      the concept meaning"@en ;
.
:itemB a skos:Concept ;
  skos:inScheme :ControlledList ;
  rdfs:label "Item B" ;
  skos:prefLabel "Item B"@en ;
  skos:definition "Description fo
      the concept meaning"@en ;
.
```

Listing 65: Concept instantiation in Turtle syntax

```
<skos:Concept rdf:about="http://base.uri/itemA">
  <skos:inScheme
      rdf:resource="http://base.uri/ControlledList"/>
  <rdfs:label>Item A</rdfs:label>
  <skos:prefLabel xml:lang="en">Item A</skos:prefLabel>
  <skos:definition xml:lang="en">Description fo the
      concept meaning</skos:definition>
</skos:Concept>
<skos:Concept rdf:about="http://base.uri/itemB">
  <skos:inScheme
      rdf:resource="http://base.uri/ControlledList"/>
  <rdfs:label>Item B</rdfs:label>
  <skos:prefLabel xml:lang="en">Item B</skos:prefLabel>
  <skos:definition xml:lang="en">Description fo the
      concept meaning</skos:definition>
</skos:Concept>
```

Listing 66: Concept instantiation in RDF/XML syntax

Rule 29 (Enumeration – in reasnoning layer)**.** For an UML enumeration, specify an equivalent class restriction covering the set of individuals that are `skos:inScheme` of this enumeration.

```
:ControlledList a owl:Class ;
  rdfs:subClassOf skos:Concept ;
  owl:equivalentClass [
    rdf:type owl:Restriction ;
    owl:hasValue :ControlledList ;
    owl:onProperty skos:inScheme ;
  ] ;
.
```

Listing 67: In-scheme equivalent class in Turtle syntax

```
<owl:Class rdf:about="http://base.uri/ControlledList">
  <rdfs:subClassOf
      rdf:resource=".../02/skos/core#Concept"/>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty
    rdf:resource=".../02/skos/core#inScheme"/>
      <owl:hasValue
          rdf:resource="http://base.uri/ControlledList"/>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Listing 68: In-scheme equivalent class in RDF/XML syntax

5 Transformation of UML descriptors

In this section are specified transformation rules for UML descriptive elements. Table 5 provides an overview of the section coverage.

UML element	Rules in core ontology layer	Rules in data shape layer	Rules in reasoning layer
NAme	Rule 30	Rule 30	Rule 30
Note	Rule 31	Rule 31	Rule 31
Comment	Rule 32	Rule 32	Rule 32

Table 5: Overview of transformation rules for UML datatypes

5.1 Name

Most of the UML elements are named. The UML conventions [7] dedicate an extensive section to the naming conventions and how deterministically to generate an URI and a label from the UML element name. The label should be associated to the resource URI by `rdfs:label` and, even if redundant, also as `skos:prefLabel`.

Rule 30 (Label). Specify a label for UML element.

```
:ResourceName
  rdfs:label "Resource name" ;
  skos:prefLabel "Resource
      name"@en ;
.
```

Listing 69: Labels in Turtle syntax

```
<rdf:Description
    rdf:about="http://base.uri/ResourceName">
  <rdfs:label>Resource name</rdfs:label>
  <skos:prefLabel xml:lang="en">Resource
      name</skos:prefLabel>
</rdf:Description>
```

Listing 70: Labels in RDF/XML syntax

5.2 Note

Most of the UML element foresee provisions of descriptions and notes. They should be transformed into `rdfs:comment` and `skos:definition`.

Rule 31 (Description). Specify a description for UML element.

```
:ResourceName
  rdfs:comment "Description of teh
      resource meaning" ;
  skos:definition "Resource name"@en ;
.
```

Listing 71: Description in Turtle syntax

```
<rdf:Description
    rdf:about="http://base.uri/ResourceName">
  <rdfs:comment>Description of teh resource
      meaning</rdfs:comment>
  <skos:definition
      xml:lang="en">Description of teh
      resource meaning</skos:definition>
</rdf:Description>
```

Listing 72: Description in RDF/XML syntax

5.3 Comment

In accordance with [5], every kind of UML Element may own Comments (see Figure 19). They add no semantics but may represent information useful to the reader. In OWL it is possible to define the annotation axiom for OWL Class, Datatype, ObjectProperty, DataProperty, AnnotationProperty and NamedIndividual. The textual explanation added to UML Class is identified as useful for conceptual modelling [3], therefore the Comments that are connected to UML Classes are taken into consideration in the transformation.

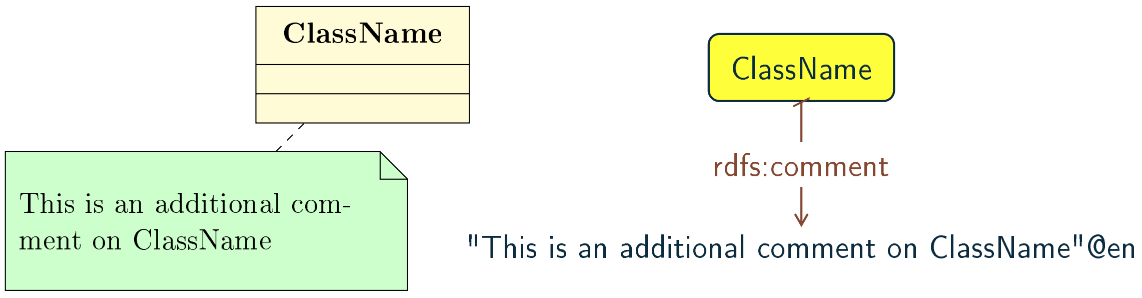

ClassName
This is an additional comment on ClassName
ClassName
rdfs:comment
"This is an additional comment on ClassName"@en

Figure 19: Visual representation of an UML comment (on the left) and an OWL comment (on the right)

As UML Comments add no semantics, they are not used in any method of semantic validation. In OWL the AnnotationAssertion [15] axiom does not add any semantics either, and it only improves readability.

Rule 32 (Comment)**.** Specify annotation axiom for UML Comment associated to an UML element.

```
:ClassName
  rdfs:comment "This is an additional
      comment on ClassName" ;
  skos:editorialNote "This is an
      additional comment on
      ClassName"@en ;
.
```

Listing 73: Comment in Turtle syntax

```
<rdf:Description
    rdf:about="http://base.uri/ClassName">
  <rdfs:comment>This is an additional
      comment on ClassName</rdfs:comment>
  <skos:editorialNote xml:lang="en">This is
      an additional comment on
      ClassName</skos:definition>
</rdf:Description>
```

Listing 74: Comment in RDF/XML syntax

Bibliography

[1] C. Atkinson and K. Kiko. A detailed comparison of uml and owl, June 2005. URL `https://madoc.bib.uni-mannheim.de/1898/`.

[2] D. Beckett. RDF/xml syntax specification (revised). W3C recommendation, W3C, Feb. 2004. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[3] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005. ISBN 0321267974.

[4] G. Carothers and E. Prud'hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.

[5] S. Cook, C. Bock, P. Rivett, T. Rutt, E. Seidewitz, B. Selic, and D. Tolbert. Unified modeling language (UML) version 2.5.1. Standard formal/2017-12-05, Object Management Group (OMG), Dec. 2017. URL `https://www.omg.org/spec/UML/2.5.1`.

[6] E. Costetchi. eProcurement ontology architecture and formalisation specifications. Recommendation, Publications Office of the European Union, April 2020.

[7] E. Costetchi. eProcurement uml conceptual model conventions. Recommendation, Publications Office of the European Union, April 2020.

[8] O. El Hajjamy, K. Alaoui, L. Alaoui, and M. Bahaj. Mapping uml to owl2 ontology. *Journal of Theoretical and Applied Information Technology*, 90(1): 126, 2016.

[9] R. Falco, A. Gangemi, S. Peroni, D. Shotton, and F. Vitali. Modelling owl ontologies with graffoo. In *European Semantic Web Conference*, pages 320–325. Springer, 2014.

[10] A. H. Khan and I. Porres. Consistency of uml class, object and statechart diagrams using ontology reasoners. *Journal of Visual Languages & Computing*, 26:42–65, 2015.

[11] A. H. Khan, I. Rauf, and I. Porres. Consistency of uml class and statechart diagrams with state invariants. In *MODELSWARD*, pages 14–24, 2013.

[12] N. Loutas, N. Loutas, S. Kotoglou, and D. Hytiroglou. D04.07 - report on policy support for eprocurement. Deliverable SC245DI07171, ISA programme of the European Commission, 2016.

[13] A. Miles and S. Bechhofer. SKOS simple knowledge organization system reference. W3C recommendation, W3C, Aug. 2009. http://www.w3.org/TR/2009/REC-skos-reference-20090818/.

[14] H.-S. Na, O.-H. Choi, and J.-E. Lim. A method for building domain ontologies based on the transformation of uml models. In *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, pages 332–338. IEEE, 2006.

[15] B. Parsia, P. Patel-Schneider, and B. Motik. OWL 2 web ontology language structural specification and functional-style syntax (second edition). W3C recommendation, W3C, Dec. 2012. http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

[16] M. Sadowska and Z. Huzar. Representation of uml class diagrams in owl 2 on the background of domain ontologies. *e-Informatica*, 13(1):63–103, 2019.

[17] G. Schreiber and F. Gandon. RDF 1.1 XML syntax. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.

[18] D. Wood, R. Cyganiak, and M. Lanthaler. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, Feb. 2014. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[19] Z. Xu, Y. Ni, L. Lin, and H. Gu. A semantics-preserving approach for extracting owl ontologies from uml class diagrams. In *International Conference on Database Theory and Application*, pages 122–136. Springer, 2009.

[20] Z. Xu, Y. Ni, W. He, L. Lin, and Q. Yan. Automatic extraction of owl ontologies from uml class diagrams: a semantics-preserving approach. *World Wide Web*, 15(5-6):517–545, 2012.

[21] J. Zedlitz and N. Luttenberger. Transforming between uml conceptual models and owl 2 ontologies. In *Terra Cognita ISWC*, pages 15–26, 2012.

[22] J. Zedlitz and N. Luttenberger. Conceptual modelling in uml and owl-2. *International Journal on Advances in Software*, 7(1):182–196, 2014.