From UML to OWL 2

Jesper Zedlitz[1], Jan Jörke[2], and Norbert Luttenberger[2]

[1] German National Library of Economics (ZBW)
[2] Christian-Albrechts-Universität zu Kiel

Abstract. In this paper we present a transformation between UML class diagrams and OWL 2 ontologies. We specify the transformation on the M2 level using the QVT transformation language and the meta-models of UML and OWL 2. For this purpose we analyze similarities and differences between UML and OWL 2 and identify incompatible language features.

1 Introduction

A class model following the Unified Modeling Language (UML) specification [11] often serves the purpose to express the conceptual model of information systems. UML's graphical syntax helps to understand the resulting models—also for non-computer scientists. Due to the rich software tool support UML class models lend themselves well to be used as a decisive artifact in the design process of information systems. The retrieval of data from information systems benefits from semantic knowledge. For this purpose it is desirable to provide an ontology for data. The commonly used language to describe ontologies is the Web Ontology Language (OWL).[15] Both modeling languages are accepted standards and have benefits in their areas of use. To leverage both languages' strengths it is usually necessary to repeat the modeling process for each language. This effort can be avoided by a transformation of a model from one language into the other.

This paper shows a transformation from a UML class model into an OWL 2 ontology by using OMG's Query/View/Transformation (QVT) [12] transformation language in conjunction with the meta-models for UML and OWL 2.

The paper is organized as follows: In section 2 we show some existing work on the transformation of UML and OWL. Section 3 explains our approach in general. Section 4 shows differences between UML and OWL 2. In section 5 we present our transformations en detail. Section 6 concludes and points out fields of future work.

2 Existing Work

A generic comparison of the seeming difference between models and ontologies is given in [1]. Differences between UML class models and OWL ontologies have been studied in [7] and [9]. [5] contains an analysis of approaches for the transformation between UML and ontologies that have been published until 2003. Transformations from UML to OWL can be grouped into three categories:

Extension of UML: [2] presents an extension of UML to improve the description of (DARPA Agent Markup Language based) ontologies using UML. [14] presents a UML based graphical representation of OWL extended by OWL annotations.

XSLT based approaches: In [6] the transformation of a UML class diagram into an OWL ontology using Extensible Stylesheet Language Transformation (XSLT) is described. Additionally a UML profile is used to model specific aspects of ontologies.

Meta-model based approaches: [4] describes a meta-model for OWL based on Meta Object Facility (MOF) and a UML profile to model ontologies using UML. [3] is a preparation for the Ontology Definition Meta-model (ODM) specification. It presents a meta-model for OWL as well as a UML profile. [10] gives a transformation between a UML model and an OWL ontology using the Atlas Transformation Language (ATL). [8] uses MOFScript to perform a UML→OWL 2 transformation. However, their goal is the validation of meta-models. Therefore they insert several model elements into the ontology that are needed for this goal but complicate the usability of the ontology.

3 Our Approach

The goal of our work is not primary to create better OWL 2 ontologies from a UML class model but to provide a well-arranged transformation that can be used for further investigation on the transformations. Each of the mentioned approaches has one or more drawbacks—e.g. syntax-dependency, no declarative transformation. Therefore we combine several thoughts from the existing approaches and apply them to the up-to-date OWL 2 Web Ontology Language (OWL 2). OWL 2 offers several new model elements that facilitate the transformation. We will highlight these elements at the relevant points. These are the five fundamental ideas of our work:

Restrict UML class models: Because we are interested in the structural schema of an information system we do not have to take the behavioral schema into account.

Meta-models on M2 level: Instead of transforming elements of a M1-model[1] directly (like an XSLT-based transformation that works with the concrete syntax of M1-models) we describe the transformation using elements of the M2 meta-

[1] A four-layer meta-model hierarchy is used for UML and MOF: M0 = Run-time instances, M1 = user model, M2 = UML, M3 = MOF.[11, sec. 7.12].

models. Thus the transformation does not depend on the model instances. It only depends on the involved meta-models.

Abstract syntax instead of concrete syntax: By working with the abstract syntax our transformation becomes independent of any particular representation (like Functional-Style Syntax, Turtle Syntax, OWL/XML Syntax, Manchester Syntax, etc.)

Declarative language: Instead of processing the input model step by step we can describe how to map elements of one model on corresponding elements of the other language.

Well-known transformation language: We choose OMG's QVT Relations Language because it is declarative and works with MOF-based meta-models. The support by the OMG consortium and several independent implementations makes it future-proof.

4 UML and OWL 2

A complete transformation between a UML class model and an OWL 2 ontology is probably not possible. There are concepts in UML that do not exist in OWL. Also the semantic of some concepts differ which hinders a transformation. [1], [7] and [9] give detailed comparisons between UML and OWL. In the following we take a look at some of these differences and show possible solutions for a transformation.

Global Names: In UML it is specified that each model element has a unique name within its package. That assures that each model element is identified by its name and its position in the package hierarchy. A similar problem applies to names of class-dependent attributes. In UML the names of these attributes have to be unique only within the class they belong to. In contrast, in OWL all names are global. Therefore we have to take care to assign each model element with a name—in OWL called "Internationalized Resource Identifier" (IRI)—that is globally unique. These IRIs are generated during the transformation process.

Unique Name Assumption: UML uses a Unique Name Assumption (UNA) which states that two elements with different names are treated as different. OWL does not use a UNA. Therefore we have to explicitly mark elements as being different.

Open-World vs. Closed-World Assumption: In UML class models we work under a Closed-World Assumption: All statements that have not been mentioned explicitly are false. In contrast OWL uses an Open-World Assumption (OWA) where missing information is treated as undecided. These different semantics make it necessary to add various restrictions to the ontology during the transformation process from a UML model to an OWL ontology to preserve the original semantics of the model. We will highlight these problems later.

Profiles: UML has the concept of "profiles" which allow extensions of meta-model elements. There is no corresponding construct in OWL 2. In most cases UML profiles are used to define stereotypes to extend classes. The information of these stereotypes can be mapped to OWL by clever creation of some new classes and generalization assertions. However a large part of an UML profile is too specific and would require transformation rules adapted for the particular profile.

Abstract Classes: Abstract classes can not be transformed into OWL 2. If a class is defined as abstract in UML no instances of this class (objects) can be created. In contrast OWL has no language feature to specify that a class must not contain any individual. An approach to preserve most of the semantics of an abstract class would be the usage of a `DisjointUnion`. This would ensure that any individual belonging to a subclass would also belong to the abstract superclass. However, it does not prohibit to create direct members of the abstract superclass.

5 Transformations

5.1 Classes

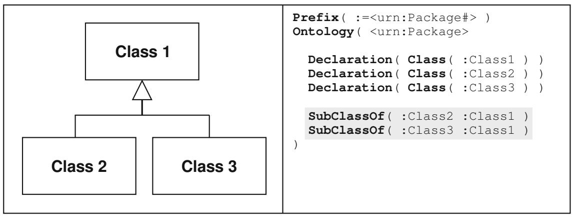
The concepts of instances resp. individuals belonging to classes exists in both UML, and OWL 2. Because both concepts are similar a basic transformation can be done with little effort. As mentioned above the missing UNA in OWL 2 makes it necessary to insert a `DisjointClasses` axiom to the ontology for every pair of classes that are not in a generalization relation.

5.2 Specialization/Generalization

The concepts of specialization and generalization in UML and OWL 2 are similar. If C' is a specialized sub-class of class C and i is an instance resp. individuum then in both cases we have $C'(i) \rightarrow C(i)$.

Therefore the transformation is straightforward. For each generalization relationship "C is generalization of C'" (resp. "C' is specialization of C") we add a `SubClassOf(` C' C `)` axiom to the ontology.

You can specify two kinds of constraints for generalizations in UML: completeness and disjointness. In a UML class diagram you can define these constraints

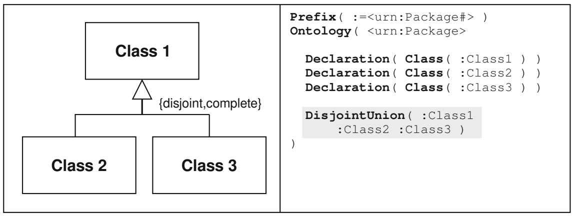


```
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>

  Declaration( Class( :Class1 ) )
  Declaration( Class( :Class2 ) )
  Declaration( Class( :Class3 ) )

  SubClassOf( :Class2 :Class1 )
  SubClassOf( :Class3 :Class1 )
)
```

Fig. 1. A generalization relationship without constraints

by adding the keywords *disjoint* or *complete* near the arrowhead of the generalization arrow. A generalization without these keywords is not subjected to the two contains.

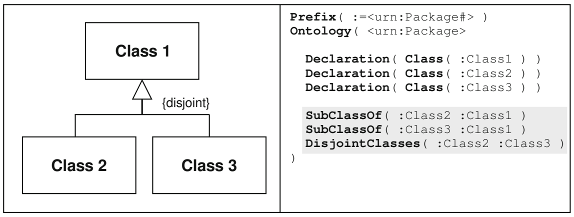
A generalization is disjoint if an instance of one sub-class must not be instance of another sub-class of the generalization. We transform this kind of constraint by adding a `DisjointClasses` axiom with all sub-classes to the ontology. This benefits from the fact that in OWL 2 a `DisjointClasses` axiom can make statements about more than two classes.



```
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>

  Declaration( Class( :Class1 ) )
  Declaration( Class( :Class2 ) )
  Declaration( Class( :Class3 ) )

  DisjointUnion( :Class1
      :Class2 :Class3 )
)
```

Fig. 2. A disjoint (but not necessary complete) generalization

If a generalization is disjoint and complete we can use a stronger axiom. The `DisjointUnion(` C C'_1 ... C'_n`)` axiom states that each individual of C is an individual of exactly one C_i and each individual of C_i is also an individual of C. The `DisjointUnion` axiom is new in OWL 2. Although it is just syntactic sugar it makes the resulting ontology easier to read.



```
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>

  Declaration( Class( :Class1 ) )
  Declaration( Class( :Class2 ) )
  Declaration( Class( :Class3 ) )

  SubClassOf( :Class2 :Class1 )
  SubClassOf( :Class3 :Class1 )
  DisjointClasses( :Class2 :Class3 )
)
```

Fig. 3. A disjoint and complete generalization

5.3 Associations and Class-Dependent Attributes

In UML classes can be connected by associations and/or class-dependent attributes. In the UML meta-model both connection kinds are represented by the model element *Property*. Therefore it stands to reason that the transformation of both associations and attributes can be handled together.

Since the model element *Association* is a subclass of *Classifier* all associations in a UML class diagram are direct members of a package. Therefore an OWL 2 concept that is similar to an association is an object property that is also direct members of an ontology. Associations can be uni- or bi-directional. One directed association can be transformed into one object property. For a bi-directional association two object properties will be created—one for each direction. To preserve the information that both resulting object properties were part of one association an `InverseObjectProperties` axiom is added to the ontology.

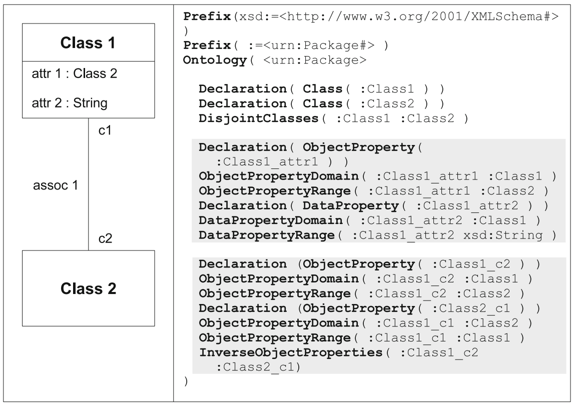
The transformation of class-dependent attributes is more complex. There are no directly corresponding concepts in OWL 2 that allow a simple transformation. The main problem is that classes in OWL 2 do not contain other model elements which would be necessary for a direct transformation. The most similar concepts in OWL 2 for class-dependent attributes are object properties and data properties.

In both cases the decision whether a *Property* is transformed into an object property or a data property depends on the *type*-association of the *Property*: If it is associated with an instance of *Class*, an object property is needed. If it is associated with an instance of *DataType*, a data property is needed.

In OWL 2 properties do not need to have a specified domain or range. In that case domain and range are implicitly interpreted as `owl:Thing`. This corresponds to the OWA: if no further information is known about a relation it might exists between individuals of any two classes. However, to limit the properties similar to the closed-world assumption of a UML class diagram we have to add the appropriate domain and range axioms that lists the allowed classes and datatypes.

The range of a property can be determined easily: It is the name of the *Classifier* linked via the *type*-association of the *Property*. For the domain we have to distinguish between properties of class-dependent attributes and those of associations. For class-dependent attributes the *class*-association of a *Property* is set. That class is the needed domain. If the *Property* is part of an association (*association*-association of the *Property* is set) we have to choose the *type*-association of the other member end's property as domain.

The OWA allows to interpret two properties that have been transformed from distinct UML properties as one. To avoid that and to map UML's CWA best we



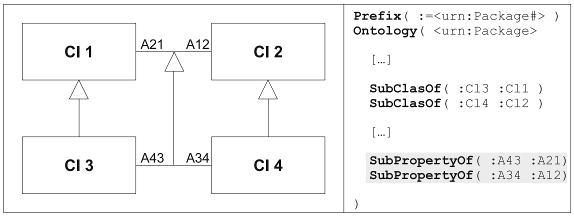
```
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>
)
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>

  Declaration( Class( :Class1 ) )
  Declaration( Class( :Class2 ) )
  DisjointClasses( :Class1 :Class2 )

  Declaration( ObjectProperty(
    :Class1_attr1 ) )
  ObjectPropertyDomain( :Class1_attr1 :Class1 )
  ObjectPropertyRange( :Class1_attr1 :Class2 )
  Declaration( DataProperty( :Class1_attr2 ) )
  DataPropertyDomain( :Class1_attr2 :Class1 )
  DataPropertyRange( :Class1_attr2 xsd:String )

  Declaration (ObjectProperty( :Class1_c2 ) )
  ObjectPropertyDomain( :Class1_c2 :Class1 )
  ObjectPropertyRange( :Class1_c2 :Class2 )
  Declaration (ObjectProperty( :Class2_c1 ) )
  ObjectPropertyDomain( :Class1_c1 :Class2 )
  ObjectPropertyRange( :Class1_c1 :Class1 )
  InverseObjectProperties( :Class1_c2
    :Class2_c1)
)
```

Fig. 4. Transformation of attributes and associations into object- and data properties

mark all properties that are not in a generalization relationship (i.e. a `SubPropertyOf` axiom exists for them) as disjoint. To do this we add OWL 2's `DisjointObjectProperties` and `DisjointDataProperties` axioms to the ontology: For all pairs of UML *Property* elements we check if they were transformed into an OWL 2 property, are not identical, no generalization relationship exists between them, and they have not been marked disjoint before.

5.4 Association Inheritance

In UML class models it is not only possible to use generalization for classes but also for associations. One association can inherit from anther association. This can be transfered to the OWL 2 world using `SubPropertyOf` axioms. Since a bi-directional association is transformed into two `ObjectProperty` axioms an inheritance relation between two associations is also transformed into two `SubPropertyOf` axioms.



```
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>

  […]

  SubClasOf( :Cl3 :Cl1 )
  SubClasOf( :Cl4 :Cl2 )

  […]

  SubPropertyOf( :A43 :A21)
  SubPropertyOf( :A34 :A12)

)
```

Fig. 5. Transformation of generalized associations

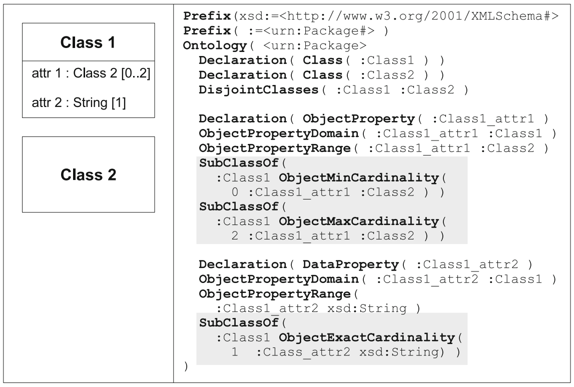
5.5 Cardinalities

In UML cardinalities can be added to associations and class-dependent attributes. A minimal and/or maximal occurrence can be a specified. OWL 2 also contains six different cardinality axioms for `ObjectProperty` and `DataProperty`: `ObjectMinCardinality`, `ObjectMaxCardinality`, and `ObjectExactCardinality` as well as the respective axioms for a `DataProperty`.

Since an `ExactCardinality` axiom is just a abbreviation of a pair of `MinCardinality` and `MaxCardinality` axioms with the same number of occurrences a UML cardinality with an equal upper and lower limit can be transformed into an `ExactCardinality` axiom to make the ontology more readable. In the case that the upper limit is 1 the property can be marked functional by adding a `FunctionalObjectProperty` (resp. `FunctionalDataProperty`) axiom to the ontology.

It is not enough just to add cardinality axioms to the ontology. In UML a class is restricted by the use of cardinalities for its class-dependent attributes and associations. However, in OWL 2 properties are not contained in classes and therefore do not restrict them by their cardinality axioms. This problem can (partly) be solved by adding `SubClassOf` axioms to the ontology that create "virtual parent classes". The only purpose of these "virtual parent classes" is to inherit their cardinality restrictions to the actual classes involved in the associations and class-dependent attributes.

A reasoner can use these cardinality restrictions to check the consistency of the ontology. However, it is only possible to detect a violation of an upper limit. A violation of a lower limit it not possible to detect due to the OWA—there might be other individuals that are simply not listed in the ontology.



```
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>
  Declaration( Class( :Class1 ) )
  Declaration( Class( :Class2 ) )
  DisjointClasses( :Class1 :Class2 )

  Declaration( ObjectProperty( :Class1_attr1 )
  ObjectPropertyDomain( :Class1_attr1 :Class1 )
  ObjectPropertyRange( :Class1_attr1 :Class2 )
  SubClassOf(
    :Class1 ObjectMinCardinality(
      0 :Class1_attr1 :Class2 ) )
  SubClassOf(
    :Class1 ObjectMaxCardinality(
      2 :Class1_attr1 :Class2 ) )

  Declaration( DataProperty( :Class1_attr2 )
  ObjectPropertyDomain( :Class1_attr2 :Class1 )
  ObjectPropertyRange(
    :Class1_attr2 xsd:String )
  SubClassOf(
    :Class1 ObjectExactCardinality(
      1  :Class_attr2 xsd:String) )
)
```

Fig. 6. Transformation of attributes and associations with cardinality constraints

5.6 Aggregation and Composition

Aggregation and *Composition* in UML are special kinds of associations between classes. There are a few restrictions for aggregations:

a) An object must not be in an aggregation association with itself.
b) An object must not be part of more than one composition.
c) An object of a class that is part of a composition must not exist without the class it belongs to.

The restriction (a) can be transformed to OWL 2 by adding an `IrreflexiveObjectProperty` axiom to the ontology. This axiom prohibits the individual to be connected to itself by the object property that represents the aggregation.

Restriction (b) can be enforced with a `FunctionalObjectProperty` or `InverseFunctionalObjectProperty` axiom. If the composition association is navigable bi-directionally the user is free to choose. Otherwise the following rules apply: If the association is navigable from 'part' to 'whole' a `FunctionalObjectProperty` is required. A connection from an individual of the 'part' class to more than one individual of the 'whole' class would make the ontology inconsistent. An `InverseFunctionalObjectProperty` is required if the association if navigable from 'whole' to 'part'.

The OWA makes a detection of (c) impossible—the individual might be part of a composition that is simply not listed in the ontology.

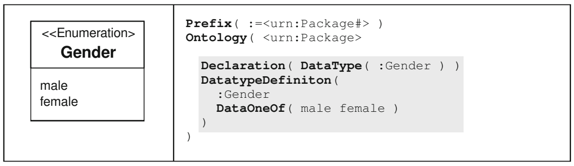
5.7 Datatypes

For the transformation of datatypes two cases have to be considered:

a) The datatype is one of the primitive UML datatypes like “Boolean”, “Integer”, or “String”. OWL 2 uses datatype definitions provided by the XML Schema specification [16]. It is transformed into its corresponding datatype from XML Schema. The indicator that a datatype belongs to this category is its package name. For primitive UML datatypes the package name is “UMLPrimitiveTypes”. In our implementation a function returns the corresponding name of the datatype from XML Schema.
b) The definition of the (user defined) datatype is given within the class model. We define a new datatype in the ontology by adding a `DatatypeDefintion` axiom.

5.8 Enumeration

In UML an *Enumeration* is used to create a datatype with a predefined list of allowed values. Although the graphical UML representation of *Enumeration* looks similar to the representation of a class they are different elements in the meta-model. Such an *Enumeration* can be transformed into a named datatype in OWL. First we have to declare the existence of the datatype. Then a definition of the datatype can be given using a `DatatypeDefinition` axiom. The allowed values of the *Enumeration* are listed in the `DataOneOf` statement.



```
Prefix( :=<urn:Package#> )
Ontology( <urn:Package>

  Declaration( DataType( :Gender ) )
  DatatypeDefiniton(
    :Gender
    DataOneOf( male female )
  )
)
```

Fig. 7. Transformation of a classes with stereotype “Enumeration”

6 Summary and Outlook

We have developed a concept for a well-arranged transformation of a UML class model into an OWL 2 ontology that can be used for the analysis of the transformation rules itself as well as the semantics of UML and OWL 2. Our concept takes up several existing approaches and combines them. The use of QVT Relations Language enables us to describe the transformation declaratively and to use model elements of the meta-models. Therefore we do not depend on the concrete syntax.

We have analyzed similarities and differences on UML elements and OWL 2 elements in-depth. With this knowledge we have developed rules to transform UML class models into elements of an OWL 2 ontology. The developed implementation of an OWL 2 meta-model conforms to the specification of [15].

Currently we are working on the opposite direction, a transformation from an OWL 2 ontology into a UML class model. That will enable us to perform a better analysis of the transformations and the semantics by comparing the results of a bi-directional transformation.

References

1. Atkinson, Guthei, Kiko: On the Relationship of Ontologies and Models. In: Proceedings of the 2nd Workshop on Meta-Modeling and Ontologies, pp. 47–60. Gesellschaft für Informatik, Bonn (2006)
2. Baclawski, K., Kokar, M.K., Kogut, P.A., Hart, L., Smith, J., Holmes III, W.S., Letkowski, J., Aronson, M.L.: Extending UML to Support Ontology Engineering for the Semantic Web. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 342–360. Springer, Heidelberg (2001)
3. Brockmans, S., Colomb, R.M., Haase, P., Kendall, E.F., Wallace, E.K., Welty, C., Xie, G.T.: A Model Driven Approach for Building OWL DL and OWL Full Ontologies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 187–200. Springer, Heidelberg (2006)
4. Brockmans, S., Volz, R., Eberhart, A., Löffler, P.: Visual Modeling of OWL DL Ontologies Using UML. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 198–213. Springer, Heidelberg (2004)
5. Falkovych, Sabou, Stuckenschmidt: UML for the Semantic Web: Transformation-Based Approaches. Knowledge Transformation for the Semantic Web 95, 92–107 (2003)
6. Gasevic, Djuric, Devedsic, Damjanovic: Converting UML to OWL Ontologies. In: Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, pp. 488–489. ACM (May 2004)
7. Hart, Emery, Colomb, Raymond, Taraporewalla, Chang, Ye, Kendall, Dutra: OWL Full and UML 2.0 Compared (March 2004)
8. Höglund, Khan, Lui, Porres: Representing and Validating Metamodels using the Web Ontology Language OWL 2, TUCS Technical Report (May 2010)
9. Kiko, Atkinson: A Detailed Comparison of UML and OWL. Technischer Bericht 4, Dep. for Mathematics and C.S., University of Mannheim (2008)
10. Milanovic, Gasevic, Giurca, Wagner, Devedzic: On Interchanging Between OWL/SWRL and UML/OCL. In: Proceedings of 6th OCLApps Workshop at the 9th ACM/IEEE MoDELS, pp. 81–95 (October 2006)
11. Object Management Group: OMG Unified Modeling Language Infrastructure, version 2.3 (2010), `http://www.omg.org/spec/UML/2.3/Infrastructure`
12. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification 1.0 (2008), `http://www.omg.org/spec/QVT/1.0/`
13. Olivé: Conceptual Modeling of Information Systems (2007) ISBN 978-3-540-39389-4
14. Schreiber: A UML Presentation Syntax for OWL Lite (April 2002)
15. World Wide Web Consortium: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (October 2009)
16. World Wide Web Consortium: XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes (December 2009)