Visual Modeling of OWL DL Ontologies Using UML

Sara Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler

Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe
{brockmans, volz, eberhart, loeffler}@aifb.uni-karlsruhe.de
http://www.aifb.uni-karlsruhe.de/WBS/

Abstract. This paper introduces a visual, UML-based notation for OWL ontologies. We provide a standard MOF2 compliant metamodel which captures the language primitives offered by OWL DL. Similarly, we invent a UML profile, which allows to visually model OWL ontologies in a notation that is close to the UML notation. This allows to develop ontologies using UML tools. Throughout the paper, the significant differences to some earlier proposals for a visual, UML-based notation for ontologies are discussed.

1 Introduction

An ontology defines a common set of concepts and terms that are used to describe and represent a domain of knowledge. Recently, several standardization committees have taken up research results form the AI community and defined standard ontology languages. For example, the Word Wide Web Consortium (W3C) has recently finished its work on the Web Ontology Language (OWL) [5].

The utility of a visual syntax for modelling languages has been shown in practice and visual modelling paradigms such as the Entity Relationship (ER) model or the Unified Modelling Language (UML) are used frequently for the purpose of conceptual modeling. Consequently the necessity of a visual syntax for knowledge representation (KR) languages has been argued frequently in the past [7,14]. Particular KR formalisms such as conceptual graphs [19] or Topic Maps [1] are based on well-defined graphical notations.

Description logic based ontology languages such as OWL, however, are usually defined in terms of an abstract (text-based) syntax and most care is spent on the formal semantics. The absence of a visual syntax[1] has lead to several proposals. [7] proposed a particular visual notation for the CLASSIC description logic. Newer developments have abandoned the idea of a proprietary syntax and proposed to rely on UML class diagrams. [4] suggested to directly use UML as an ontology language, whereas [2] proposed to predefine several stereotypes such that a more detailed mapping from UML to the primitives offered by the DAML+OIL description logic can be achieved. [2] further argue that the UML

[1] Which can be seen as a direct result of the criticisms about the semantics of early diagrammatic semantic networks [21,3].

S.A. McIlraith et al. (Eds.): ISWC 2004, LNCS 3298, pp. 198–213, 2004.

metamodel should be extended with elements such as property and restriction such that UML is more compatible with KR languages such as OWL.

A metamodel for the purpose of defining ontologies, called Ontology Definition Metamodel (ODM), has recently been requested by the OMG [18], with specific focus on the OWL DL language. In answer to this request four proposals were submitted. [12] defines stereotypes for representing the new aspects introduced by ontologies in UML. In our opinion, the proposal has several weaknesses. For example, OWL properties are visually modelled as UML classes instead of mapping them to UML associations. Similarly, the different kind of class constructors found in OWL are reflected by associations, while we believe that an appropriate specialization of UML classes is more appropriate. Obviously there are no strict metrics for intuitiveness and clarity. Consequently, in this paper we argue by showing head to head comparisons of the different representations. Ultimately, the choice is up to the reader.

[8] follows the RDF serialization of OWL documents and represents the ontology in a graph-like notation that is close to the actual RDF serialization. Also various kinds of OWL properties are modeled as a hierarchy of UML classes. Complex OWL class definitions, i.e. using union and other operators, are modeled as comments associated to a class.

[13] departs from the OMG request and introduces a metamodel for the Open Knowledge Base Connectivity (OKBC) standard. [6] suggests the OWL Full language, but neither provides a visual syntax nor introduces a metamodel.

At the moment [12,6,8,13] are merged into one proposal. While this is work in progress, a first presentation [10] suggests that the proposal will be based on OWL Full and is being extended to capture further KR languages such as Simple Common Logic (SCL). Obviously the scope of this merged proposal is quite extensive.

The benefit from this design decision is that different ontology formats such as KIF, Description Logics, or Topic Maps can be mapped into one very general and expressive metamodel. Also, a single mapping to the visual UML profile in which actual ontologies are defined suffices.

However, these advantages do not come for free. By encompassing many paradigms, the resulting ODM and the mappings become quite complex. We believe that, for the sake of readability and usability, several separate metamodels should be introduced for each KR, e.g. OWL DL. In order to map between the individual KRs the requested metamodel mapping facilities [17] can be used. While this results in a higher number of mappings, the individual mappings will be a lot more lightweight and easier to specify.

Therefore this paper defines an ODM for OWL DL. Our goals are to achieve an intuitive notation, both for users of UML and OWL DL. Naturally, the proposed metamodel has a one-to-one mapping to the abstract syntax of OWL DL and thereby to the formal semantics of OWL.

The remainder of this paper is organized as follows: Section 2 introduces the Meta Object Facility (MOF). Section 3 introduces our Ontology Definition Metamodel for OWL DL. Section 4 introduces a UML Profile for ontology mod-

eling and explains the major design choices taken in order to make the notation readable for both users with UML and users with OWL background. We conclude by summarizing our work and listing points for future work.

2 UML-MOF

This section introduces the essential ideas of the Meta Object Facility (MOF) of UML 2.0 and shows how an Ontology Definition Metamodel (ODM) fits into the overall picture. The need for a dedicated ontology modeling language stems from the observation that an ontology cannot be sufficiently represented in UML [11]. Both representations share a set of core functionalities such as the ability to define classes, class relationships, and relationship cardinalities. Despite this overlap, there are many features which can only be expressed in an ontology language. Examples for this disjointness are transitive and symmetric properties in OWL or methods in UML.

UML methodology, tools and technology, however, seem to be a feasible approach for supporting the development and maintenance of ontologies. Consequently, the OMG issued a request for proposals for an Ontology Definition Metamodel (ODM) [9]. The following key requirements were given:

1. As shown in Figure 1, an ODM has to be grounded in the Meta Object Facility (MOF2), which is explained in the following section. This requirement is common for any other metamodel, e.g. UML.
2. A UML profile defining a visual notation for ontologies must be provided. Furthermore, partial mappings in both directions between the metamodel and this profile need to be established.
3. From the ODM, one must be able to generate an ontology representation in a language such as OWL DL. Figure 1 shows this process. In particular a mapping to OWL DL was requested.
4. An XMI serialization and exchange syntax for ODM must be provided. This XMI format allows exchanging an ODM metamodel between tools.

We target the first two requirements in this paper, since the remaining two requirements directly follow from a good ODM.

2.1 Meta Object Facility

The Meta Object Facility (MOF) is an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. The aim is to provide a framework that supports any kind of metadata, and that allows new kinds to be added as required. MOF plays a crucial role in OMG's four-layer metadata architecture shown in Figure 2. The bottom layer of this architecture encompasses the raw information to be described. For example, Figure 2 contains information about a person called Pete and a car with the license plate ABC-1234. The model layer contains the

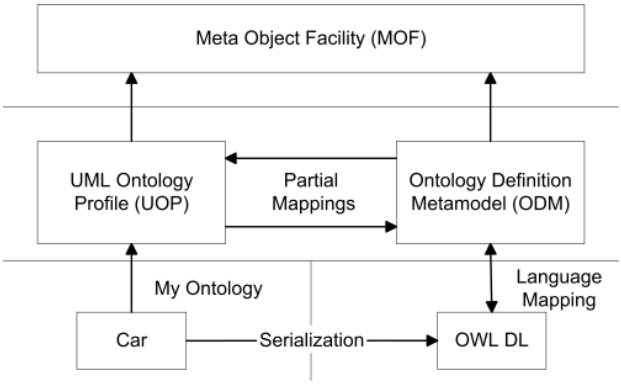


Fig. 1. A partial ontology to UML mapping allows existing tools to operate on compatible aspects of ontologies.

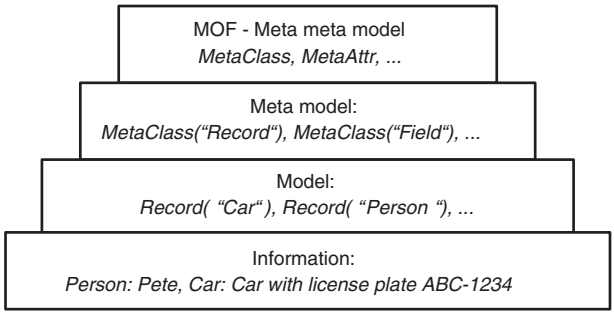


Fig. 2. OMG Four Layer Metadata Architecture.

definition of the required structures. Our domain might use records for grouping information. Consequently, the records car and person are defined. If these are combined, they describe the model for the given domain. The metamodel defines the terms in which the model is expressed. In our example, we would state that models are expressed with records and fields by instantiating the respective meta classes. Finally, the MOF lies at the top. This layer is called the meta meta model layer. Note that MOF is hard wired, while the other layers are flexible and allow to express various metamodels such as the UML metamodel.

We provide the ODM for OWL DL in the following section using the core modeling features provided by MOF. We primarily use classes, attributes and associations, the notation of which is well-known from UML. Additionally, our metamodel is augmented with OCL constraints which specify invariants that have to be fulfilled by all models that instantiate the ODM. Such models are visually encoded using the UML profile introduced in Section 4.

3 Ontology Definition Metamodel

3.1 Design Considerations

A metamodel for a language that allows the definition of ontologies naturally follows from the modelling primitives offered by the ontology language. OWL ontologies themselves are RDF documents. They instantiate the RDF data model, and use URIs to name entities. The formal semantics of OWL is derived from Description Logics (DL), an extensively researched KR formalism. Hence, most primitives offered by OWL can also be found in a Description Logic. Three species of OWL have been defined. One variant called OWL Full can represent arbitrary RDF components inside of OWL documents. This allows, for example, to combine the OWL language with arbitrary other representation languages. From a conceptual perspective a metamodel for OWL Full necessarily has to include elements for the representation of RDF.

Another variant called OWL DL states syntactic conditions on OWL documents, which ensure that only the primitives defined within the OWL language itself can be used. OWL DL closely corresponds to the SHOIN(D) description logic and all language features can be reduced[2] to the primitives of the SHOIN(D) logic. Naturally, a metamodel for OWL DL is smaller and less complex than a metamodel for OWL Full. Similarly, a OWL DL metamodel can be built in a way such that all elements can be easily understood by people familiar with description logics. A third variant called OWL Lite disallows some constructors of OWL DL, specifically number restrictions are limited to cardinalities 0 and 1. Furthermore, the oneOf class constructor is missing. Other constructors such as class complement, which are syntactically disallowed in OWL Lite, can nevertheless be represented via the combination of syntactically allowed constructors [20][Corollary 3.4.1]. Hence, a metamodel for OWL DL necessarily includes OWL Lite.

3.2 An ODM for OWL DL

The rest of this section will provide a summary of the OWL language whilst introducing our metamodel. Interested readers may refer to the specifications [16] for a full account of OWL. The metamodel is augmented with several OCL constraints. Some important constraints are given here in footnotes.

3.2.1 Ontologies. URIs are used to identify all objects in OWL. In order to provide an efficient notation, we replicate the namespace concept of XML and introduce a separate `Namespace` metaclass which manages the abbreviation (`name`) that is assigned to a certain URI (cf. Figure 3). Every element of an ontology is a `NamedElement` and hence a member of a `Namespace`. All elements of an `Ontology` are specializations of `OntologyElement`[3] which is itself derived

[2] Some language primitives are shortcuts for combinations of primitives in the logic.

[3] `member->forAll(ocllsKindOf(OntologyElement))`

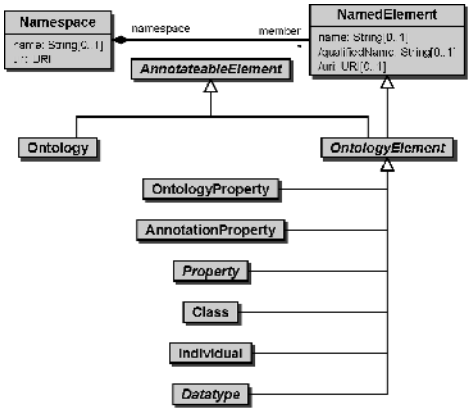


Fig. 3. Ontologies and Namespaces

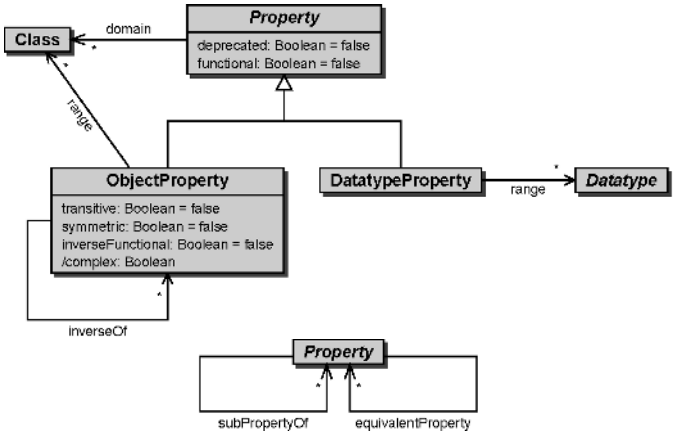


Fig. 4. Properties and property axioms

from `NamedElement`. Anonymous elements of the ontology belong to a dedicated anonymous namespace[4]. The qualifiedName attribute is the global name of an element and can be derived[5] from the local `name` attribute and the `Namespace name`.

3.2.2 Properties. Properties represent named binary associations in the modeled knowledge domain. OWL distinguishes two kinds of properties, so called object properties and datatype properties. Both are generalized by the

[4] `allInstances()->size(name->isEmpty())) <=1`

[5] `name->notEmpty() and namespace.name->notEmpty() implies qualifiedName = namespace.name.concat(":").concat(name).`

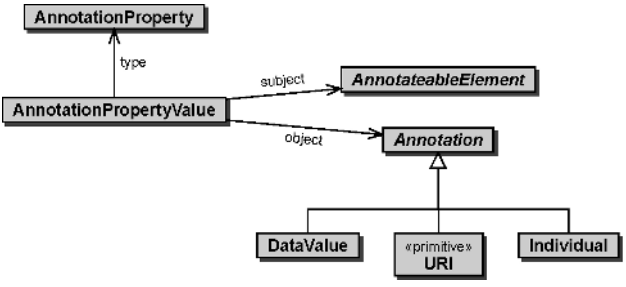


Fig. 5. Annotations

abstract metaclass `Property`. Properties can be functional, i.e. their range may contain at most one element. Their domain is always a class. Object properties may additionally be inverse functional, transitive, symmetric or inverse to another property. Their range is a class[6], while the range of datatype properties is a datatype.

Users can relate properties by using two axioms. Property subsumption (`subPropertyOf`)[7] specifies that the extension of a property is a subset of the related property. Similarly, property equivalence (`equivalentProperty`) defines extensional equivalence. OWL DL disallows that object and datatype properties are related via axioms.

3.2.3 Ontology properties. Ontologies themselves can have properties, which are represented via the `OntologyProperty` metaclass. For example, the ontology property `owl:imports` allows to logically include the elements of one ontology in another ontology. OWL DL predefines several ontology properties and allows users to define further ontology properties. A concrete instance of an ontology property is represented through `OntologyPropertyValue`, which instantiates a certain type of `OntologyProperty` and is a reference between two ontologies.

3.2.4 Annotation properties. Given elements of an OWL ontology can be annotated with metadata. Several annotation properties, e.g. `owl:versionInfo`, are predefined and users can define further annotation properties. We treat annotation properties similarly to ontology properties. However, the subject of an `AnnotationPropertyValue` is an `AnnotateableElement`

[6] OWL DL mandates that no complex role may be transitive:
`complex=functional or inverseFunctional or NumberRestriction.`
`allInstances()->exists(onProperty=self) or inverseOf->exists(complex)`
`or subPropertyOf->exists(complex)` and `complex implies not transitive`.

[7] This association is transitive:
`Property.allInstances()-> forAll(r,s,t|(r.subPropertyOf->includes(s)`
`and s.subPropertyOf->includes(t) implies r.subPropertyOf->includes(t))).`

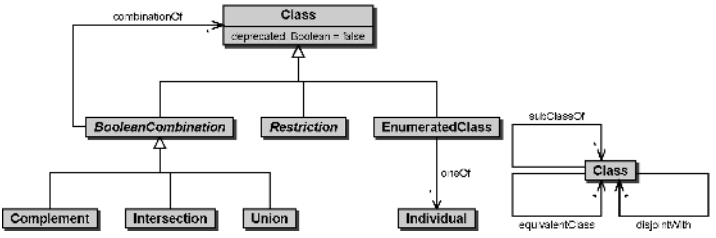


Fig. 6. Class constructors and axioms

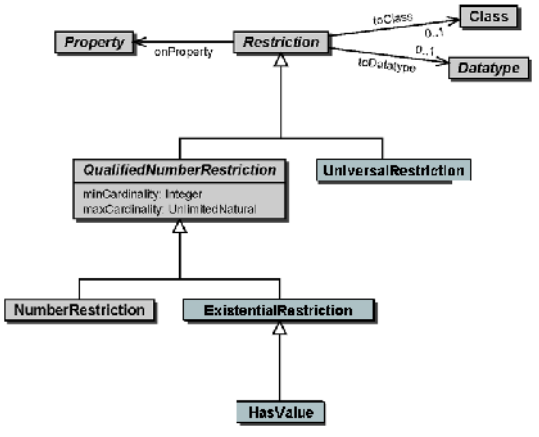


Fig. 7. OWL Restrictions

and the object is a Annotation, which can be either a `DataValue`, a `URI` or an `Individual` (cf. Figure 5).

3.2.5 Class Constructors. In comparison to UML, OWL DL does not only allow to define simple named classes. Instead, classes can be formed with several class constructors (cf. Figure 6). One can conceptually distinguish the boolean combination of classes, restrictions and enumerated classes. `EnumeratedClass` is only available in OWL DL and is defined through a direct enumeration of named[8] individuals. Boolean combinations of classes are provided through `Complement`[9], `Intersection` and `Union`.

Restrictions are class constructors that restrict the range of a property for the context of the class (cf. Figure 7). Restrictions can be stated *on* datatype and object properties, as indicated by `toClass` and `toDatatype`.

[8] `oneOf->forAll(name.notEmpty())`
[9] `combinationOf->size()=1`

Accordingly they limit the value *to* a certain datatype or class extension[10]. `UniversalRestriction` provides a form of universal quantification that restricts the range of a class to the extension of a certain class or datatype[11].

We introduce an abstract metaclass `QualifiedNumberRestriction` to relate unqualified cardinality restrictions (which are available in OWL) and existential restrictions. Obviously the minimum cardinality is by default 0 and may not be negative[12] while the maximum cardinality should not be smaller than the minimum cardinality[13]. Unqualified number restrictions (`NumberRestriction`) are available in OWL and define how many elements the range of the given property has to have while not restricting the type of the range[14]. (`ExistentialRestriction`) can logically and semantically be seen as a special type of qualified number restrictions where the cardinality is fixed[15]. OWL also provides `HasValue`, which is a special type of existential restriction where the qualifying class is an enumeration containing a single individual[16].

Figure 6 shows that classes can be related with each other using class axioms, such as class subsumption (`subClassOf`), class equivalence (`equivalentClass`)[17] and class disjointness (`disjointWith`). These relations between classes are naturally modelled as associations.

3.2.6 Datatypes. The datatype system of OWL is provided by XML Schema, which provides a predefined set of named datatypes (`PrimitiveType`), e.g. strings `xsd:string`. Additionally, users may specify enumerated datatypes (`EnumeratedDatatype`) which consist of several data value of items (`DataValue`).

3.2.7 Knowledge Base. OWL does not follow the clear conceptual separation between terminology (T-Box) and knowledge base (A-box) that is present in most description logics and in MOF, which distinguishes between model and information. The knowledge base elements (cf. Figure 8) are

[10] 1. `toClass->size()=1 xor toDatatype->size()=1`
2. `onProperty.ocllsKindOf(DatatypeProperty) implies toDatatype->size()=1`

[11] The reader may note that this is logically not understood as a constraint but as an entailment rule.

[12] `minCardinality>=0`

[13] Even though OWL allows this by making the class definition become inconsistent. We disallow this situation through the constraint:
`maxCardinality>=minCardinality`.

[14] `toClass=owl::Thing` or `toDatatype=rdfs::Literal`

[15] `minCardinality=1 and maxCardinality=*`

[16]
```
toClass.oclssTypeOf(EnumeratedClass) and
toClass.oclAsType(EnumeratedClass).oneOf->size()=1)
or (toDatatype.ocllsTypeOf(EnumeratedDatatype) and
toDatatype.oclsAsType(EnumeratedDatatype).oneOf->size()=1)
```

[17] Every equivalent class is trivially a superclass:
`subClassOf->includesAll(equivalentClass)`.

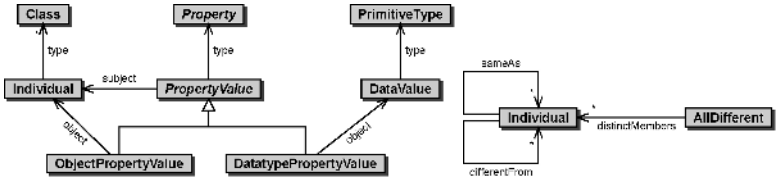


Fig. 8. Knowledge Base Items and Axioms

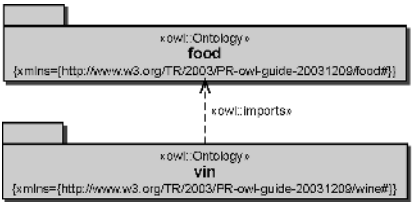


Fig. 9. owl:imports

part of an ontology. An `Individual` is an instantiation of a `Class` and is the subject of a `PropertyValue`, which instantiates a `Property`. Naturally, an `ObjectPropertyValue` relates its subject with another `Individual` whilst a `DatatypePropertyValue` relates its subject with a `DataValue`, which is an instance of a primitive datatype.

Individuals can be related via three axioms. The `sameAs` association allows users to state that two individuals (with different names) are equivalent. The `differentFrom` association specifies that two individuals are not the same[18]. `AllDifferent` is a simpler notation for the pairwise difference of several individuals.

4 A UML-Profile for Ontologies

This section describes a UML profile which supports reusing UML notation for ontology definition. Since the UML profile mechanism supports a restricted form of metamodeling, our proposal contains a set of extensions and constraints to the UML metamodel. This tailors UML such that models instantiating the ODM can be defined. We heavily rely on the custom stereotypes, which usually carry the name of the corresponding OWL language element.

4.1 Ontologies

Figure 9 shows that a `Namespace` is represented by packages, while a stereotype indicates an `Ontology`. Ontology properties correspond to appropriately stereotyped UML dependencies. The deprecation of a given element, e.g. the *deprecated class* JugWine in Figure 10, is achieved using a stereotype.

[18] The reader may note that OWL does not take the unique names assumption.

Fig. 10. owl:DeprecatedClass

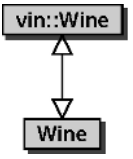
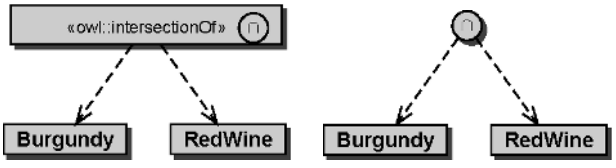


Fig. 11. owl:EquivalentClass



(a) normal notation (b) alternative notation

Fig. 12. owl:intersectionOf

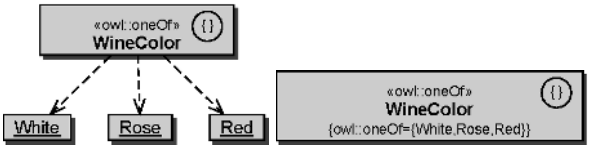
4.2 Classes

Atomic classes are depicted in the most trivial way, namely as UML classes. The reader may note, that we only use the first segment of the UML class notation, which contains the name of the class, stereotypes, and keyword-value pairs. The second segment specifies concrete properties, while the third segment is missing, since OWL does not contain methods. *Class inclusion* is depicted using the UML generalization symbol, which is the most natural way.

Class equivalence could be expressed by two class inclusions. As a simpler notation for two generalization arrows in the opposite direction next to each other, the bi-directional generalization arrow is introduced. An example of this notation is shown in Figure 11. Dependencies could also be used but are not intuitive. [12,10] propose to use stereotyped UML associations to state class axioms, which does not translate well to the UML object level. For this reason, *Class disjointness* is depicted as a bi-directional, stereotyped dependency.

For the representations of OWL class constructors, we use individual stereotypes and the UML class notation. Dependencies to the classes which form the complement, part of the union or part of the intersection of a class are depicted as UML dependencies. We suggest specific pictograms to be used instead of dependencies as allowed in UML. Figure 12 depicts alternative graphical notations for an intersection of classes. An `EnumeratedClass` is connected to the enumerated individuals by dependencies (cf. Figure 13). Alternatively, we allow a more compact string-based notation. The reader may note that UML associations can only be used between classes, an `EnumeratedClass` can therefore not be consistently represented with associations, if the UML notation for objects is used for individuals.

In general, a *restriction* is depicted by a class with a corresponding stereotype. If the property which participates in the restriction is an object property,



(a) normal notation (b) compact notation

Fig. 13. owl:oneOf

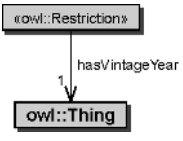
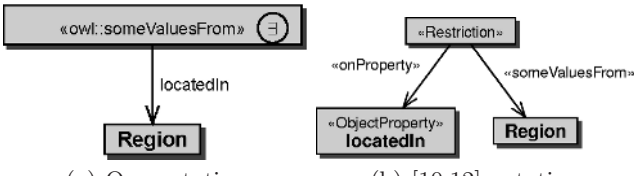


Fig. 14. owl:cardinality



(a) Our notation (b) [10,12] notation

Fig. 15. owl:someValuesFrom

we depict it as an association to the participating class. Otherwise, in case of a datatype property, it is depicted as an attribute. Figure 14 shows that cardinalities involved in restrictions are depicted in the standard UML notation, viz. next to the attribute's association. We mentioned that OWL has only unqualified cardinality restrictions. Thus, the class participating in a cardinality restriction is always owl:Thing and attribute types are rdfs::Literal, which means that they can have every data value.

`ExistentialQuantification` can and `ValueRestriction` has to be indicated by a DEDICATED stereotype. Figure 15 demostrates the notation for an existentially quantified restriction. The reader can compare our notation with the notation proposed by [12,10] (cf. Figure 15). Clearly, our presentation is more compact and elegantly uses the available features of UML.

When modeling `HasValue`, no separate notation is introduced. If properties are represented as associations, the endpoints have to be classes. Under these circumstances, combining existence restriction and enumeration is the most compact notation conforming to the UML-metamodel. One could think to model it more directly from the class which has the restriction, but an association cannot be built between a class and an individual. Although our solution looks

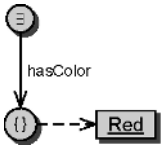


Fig. 16. owl:hasValue

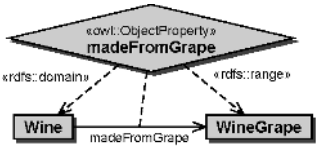


Fig. 17. An ObjectProperty with domain and range

quite complex, it keeps the consistency with restrictions. Figure 16 shows our notation.

4.3 Properties

Object properties are represented as UML n-ary associations[19], while datatype properties are represented as UML attributes. Since properties can have multiple domains and ranges, several associations with the same name are needed, therefore our proposal uses an association class which is connected to the association itself. If the domain is itself a restriction, we end up with two associations and it would be unclear which one counts for the restriction and which one for the domain of the property. In this case, we provide a extended graphical representation (cf. Figure 17).

Analogous to classes, specific properties are assigned a respective stereotype. Figure 18 demonstrates the *functionality* and *inverse functionality* stereotype. Naturally, *Deprecation*, *transitivity* and *symmetry* are represented in the same way. Figure 18 also shows how a property is connected to its inverse using a bi-directional UML dependency.

Similar to classes, *Property inclusion* is depicted with a generalization arrow, and *property equality* with a bi-directional generalization arrow.

4.4 Data Types

Data types are represented in the form of a stereotyped UML class. An `EnumeratedDatatype` is depicted similar to the enumeration of individuals, viz. a stereotyped UML class is connected to the enumerated data values through dependencies and we provide a text-based shorthand notation (cf. Figure 19).

[19] This notation of associations is in fact provided by UML although rarely seen in practice.

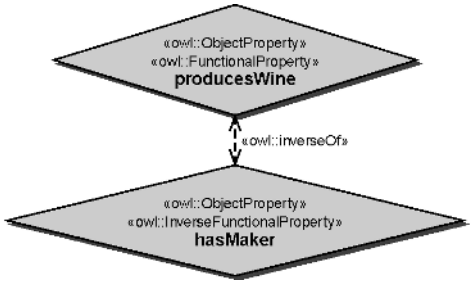
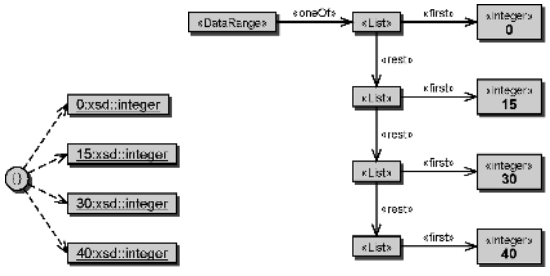


Fig. 18. Property characteristics



(a) Our notation (b) [12] notation

Fig. 19. Enumerated Datatypes

Figure 19 also shows the proposed notation by [12], which uses associations and additional stereotyped classes. Sticking to the RDF-notation makes the representation unnecessarily complex since there is no order in data enumerations.

4.5 Individuals

Individuals are depicted in the object notation of UML, viz. in the form 'Object : Class'. [12] proposes to use a Class Thing for individuals, but this does not clearly show the difference between the object and model level, which is pursued in UML. We represent axioms specifying the equivalence or difference of individuals through stereotyped associations between individuals. We conclude with Figure 20, which shows our notation for `AllDifferent`. Here, associations lead from an anonymous instance of owl::AllDifferent to those individuals which are defined to be different.

5 Conclusion

We have presented an Ontology Definition Metamodel for the DL variant of OWL. Unlike previous proposals, our metamodel directly corresponds to the

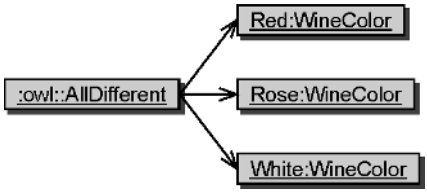


Fig. 20. owl:AllDifferent

language primitives available in OWL DL. The validity of instances of this metamodel is ensured through various OCL constraints, some of which were given here (cf. [15] for a full account). We additionally provided a UML profile, of which we believe that it is cognitively more adequate for folks familiar with UML *and* OWL. Our profile utilizes the maximal intersection of UML features and OWL features. Hence, classes are depicted as classes, properties as n-ary associations and individuals as UML objects.

We believe that leveraging UML for the development and maintenance of ontologies is a very promising approach. It is a first step to bring the W3C vision of a Semantic Web technology and the OMG vision of a Model Driven Architecture together. We can now use a large array of industrial strength tools that is available for UML and other related OMG standards for the purpose of ontology development. Besides graphical editors, other kinds of utilities offer further benefit. For example, we can utilize the Eclipse Modeling Framework (EMF) to derive a Java API for OWL directly from the ODM. Ontologies can benefit from UML based system development, but in turn ontologies can also contribute to system development. One of the prime application areas is the management of policies and the enforcement of regulatory compliance in logistics, the financial sector, or other industries. These very advantages and application areas prompted OMG's call for proposals for an ontology definition model.

Acknowledgments. Research for this paper has been partially funded by the EU in the IST projects KnowledgeWeb (IST-2004- 507482) and Sekt (IST-2003-506826), as well as by the Graduate School IME - University Karlsruhe. We would like to thank our colleagues for discussion as well as the reviewers for the ISWC conference for valuable comments on our paper.

References

1. Topic Maps: Information Technology – Document Description and Markup Languages. ISO/IEC standard 13250:2000, December 1999.
2. K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M. Aronson. Extending UML to Support Ontology Engineering for the Semantic Web. In *4th Int. Conf. on UML (UML 2001)*, Toronto, Canada, October 2001.
3. R.J. Brachman. On the Epistemological Status of Semantic Nets. In N.V. Findler, editor, *Associative Networks: Representation and Use of Knowledge by Computers*, pages 3–50, 1979.

4. S. Cranefield and M. Purvis. UML as an Ontology Modelling Language. In *Proceedings of the Workshop on Intelligent Information Integration*, volume 23 of *CEUR Workshop Proceedings*, Stockholm, Sweden, July 1999.
5. M. Dean and G. Schreiber. Web Ontology Language (OWL) Reference Version 1.0. Technical report, World Wide Web Consortium (W3C), 2003.
6. DSTC. Ontology Definition MetaModel Initial Submission. `http://www.omg.org/docs/ad/03-08-01.pdf`, August 2003.
7. B. R. Gaines. An Interactive Visual Language for Term Subsumption Languages. In J. Mylopoulos and R. Reiter, editors, *Proc. of 12th Int. Joint Conf. on Art. Int.*, pages 817–823, Sydney, Australia, August 1991. Morgan Kaufmann.
8. Gentleware. Ontology Definition Meta-Model. `http://www.omg.org/docs/ad/03-08-09.pdf`, August 2003.
9. Object Management Group. Ontology Definition Metamodel - Request For Proposal, March 2003.
10. L. Hart, P. Emery, B. Colomb, K. Raymond, D. Chang, Y. Ye, E. Kendall, and M. Dutra. Usage Scenarios and Goals For Ontology Definition Metamodel. `http://www.omg.org/docs/ontology/04-01-01.pdf`, January 2004.
11. L. Hart, P. Emery, B. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, and M. Dutra E. Kendall. OWL Full and UML 2.0 Compared, March 2004.
12. IBM. Ontology Definition Metamodel (ODM) Proposal. `http://www.omg.org/docs/ad/03-07-02.pdf`, August 2003.
13. Sandpiper Software Inc. and Stanford University Knowledge Systems Laboratory. UML for Knowledge Representation. A Layered, Component-Based Approach to Ontology Development. `http://www.omg.org/docs/ad/03-08-06.pdf`, March 2003.
14. R. Kremer. Visual Languages for Knowledge Representation. In *Proc. of 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Voyager Inn, Banff, Alberta, Canada, April 1998. Morgan Kaufmann.
15. P. Loeffler. UML zur Visuellen Modellierung von OWL DL. Master's thesis, University of Karlsruhe (TH), June 2004.
16. D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium (W3C), August 2003. Internet: `http://www.w3.org/TR/owl-features/`.
17. Object Management Group. MOF 2.0 Query / Views / Transformations - Request for Proposal. `http://www.omg.org/docs/ad/02-04-10.pdf`, 2002.
18. Object Management Group. Ontology Definition Metamodel - Request for Proposal. `http://www.omg.org/docs/ontology/03-03-01.rtf`, 2003.
19. J. F. Sowa. Conceptual Graphs Summary. In P. Eklund, T. Nagle, J. Nagle, and L. Gerholz, editors, *Conceptual Structures: Current Research and Practice*, pages 3–52, 1992.
20. R. Volz. *Web Ontology Reasoning with Logic Databases*. Phd thesis, University of Karlsruhe (TH), Karlsruhe, Germany, `http://www.ubka.uni-karlsruhe.de/cgi-bin/psview?document=2004/wiwi/2`, February 2004.
21. W.A. Woods. What's in a Link: Foundations for Semantic Networks. In D.G. Bobrow and A.M. Collins, editors, *Representation and Understanding: Studies in Cognitive Science*, pages 35–82, 1975.