A Metamodel and UML Profile for Networked Ontologies – The Complete Reference

Saartje Brockmans, Peter Haase
Institute AIFB, Universität Karlsruhe, Germany
{brockmans, haase}@aifb.uni-karlsruhe.de

Revision: May 13, 2006

Abstract

In this paper we present integrated MOF compliant metamodels and UML profiles for OWL DL, the Semantic Web Rule Language (SWRL) and Ontology Mappings. Based on these metamodels and profiles, UML tools can be used for visual modeling of rule-extended ontologies and ontology mappings.

Contents

Chapter 1

Metamodeling for Ontologies

1.1 Motivation

The metamodeling features of Model Driven Architecture (MDA) provide the means for the specification of modeling languages in a standardized, platform independent manner. In short, the Meta Object Facility (MOF, [Obj02]) provides the language for creating metamodels, UML defines the language for creating models corresponding to specific metamodels. Defining the networked ontology model in terms of a MOF compliant metamodel yields a number of advantages:

Interoperability with Software Engineering approaches In order for semantic technologies to be widely adopted by users and to succeed in real-life applications, it must be well integrated with mainstream software trends. This includes in particular interoperability with existing software tools and applications to put it closer to ordinary developers. MDA is a solid basis for establishing such interoperability. With the networked ontology model defined in MOF, we can utilize MDA's support in modeling tools, model management and interoperability with other MOF-defined metamodels.

Reuse of UML for modeling With respect to interoperability with other metamodels, UML is of particular importance. UML is a well established formalism for visual modeling and recently has been proposed as a visual notation for knowledge representation languages as well . While UML itself lacks specific features of KR languages, the extension mechanisms, UML profiles, allow to tailor the visual notation to the specific required needs.

Independence of particularities of specific formalisms The metamodeling approach of MDA and MOF allows to define the networked ontology model in an abstract form independent of the particularities of specific logical formalisms. This enables to be compatible with currently competing formalisms (e.g. in the case of mapping languages), for which no standard exists yet. Language mappings, also called groundings, define the relationship with particular formalisms and provide the semantics for the networked ontology model. Further, the extensibility capabilities of MOF allow to add new modules to the metamodel if required in the future.

1.2 Metamodeling with MOF

This section introduces the essential ideas of MOF and shows how a metamodel and a UML profile for networked ontologies fit into this more general picture. The need for a dedicated visual ontology modeling language stems from the observation that an ontology cannot be sufficiently represented in UML [HEC$^+$04]. The two representation mechanisms share a set of core functionalities such as the ability to define classes, class relationships, and relationship cardinalities. But despite this overlap, there are many features which can only be expressed in OWL, and others which can only be expressed in UML. Examples for this disjointness are transitive and symmetric properties in OWL or methods in UML. For a full account of the conceptual differences we refer the reader to [IBM05].

UML methodology, tools and technology, however, seem to be a feasible approach for supporting the development and maintenance of ontologies, rules and ontology mappings. The general idea of using MOF-based metamodels and UML profiles for this purpose is depicted in Figure 1.1 : A metamodel for rule-extended ontologies and ontology mappings as well as a UML profile are grounded in MOF, in that they are defined in terms of the MOF meta-metamodel, explained further in this section. The UML profile mechanism is an extension mechanism to tailor UML to specific application areas. Our proposed UML profile defines a visual notation for optimally supporting the specification of ontologies, rules and ontology mappings. This visual syntax is based on the metamodel and is independent from a concrete mapping formalism. Mappings in both directions between the metamodel and the profile have to be established.

OWL DL ontologies, SWRL rules and ontology mappings in a concrete language instantiate the metamodel. The constructs of the specific languages have a direct correspondence with those of the metamodel. Analogously, specific UML models instantiate the UML profile. Within the MOF framework, the UML models are translated into definitions based on the above mappings between the metamodel and the UML profile. In case of ontology mappings, in this translation step, so after the visual modeling of the ontology mappings, the decision about a concrete mapping formalism is taken, based on the types of the mappings which were modeled.

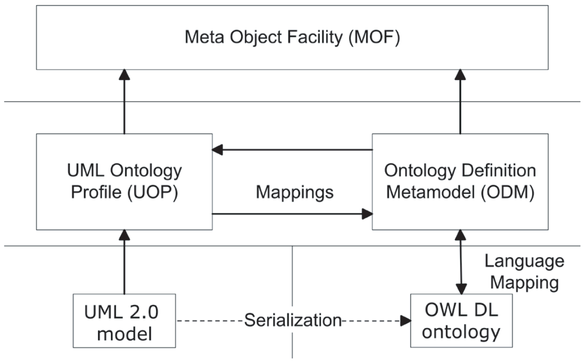
Meta Object Facility (MOF)
UML Ontology Profile (UOP)
Mappings
Ontology Definition Metamodel (ODM)
Language Mapping
UML 2.0 model
Serialization
OWL DL ontology

Figure 1.1: How a metamodel and a UML profile for ontologies fit into the picture of the Meta Object Facility framework

1.2.1 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 goal 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 the four-layer metadata architecture of the Object Management Group (OMG) shown in Figure 1.2. The bottom layer of this architecture encompasses the raw information to be described. For example, Figure 1.2 contains information about a wine called ElyseZinfandel and about the Napa region, where this wine grows. The model layer contains the definition of the required structures, e.g. in the example it contains the classes used for grouping information. Consequently, the classes Wine and Region 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 classes and properties by instantiating the respective meta classes. Finally, the MOF constitutes the top layer, also called the meta-metamodel layer. Note that the top MOF layer is hard wired in the sense that it is fixed, while the other layers are flexible and allow to express various metamodels such as the UML metamodel or the metamodel for OWL DL ontologies, SWRL rules and ontology mappings.

1.2.2 Ontology Definition Metamodels

The general idea of using MOF-based metamodels and UML profiles for the purpose of engineering ontologies is depicted in Figure 1.1 for the OWL ODM: The ODM as well as a UML profile are grounded in MOF, in that they are defined in terms of the MOF meta-metamodel, explained further in this section. The UML profile mechanism is an extension mechanism to tailor UML to specific application areas. UML profiles define a visual notation for optimally supporting

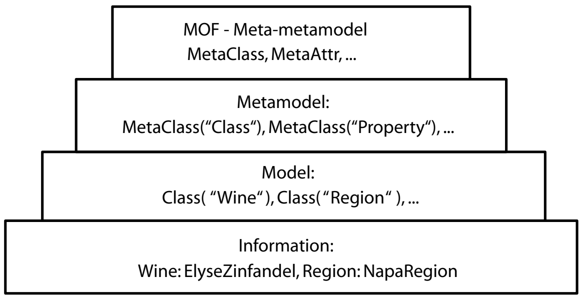
MOF - Meta-metamodel
MetaClass, MetaAttr, ...
Metamodel:
MetaClass("Class"), MetaClass("Property"), ...
Model:
Class("Wine"), Class("Region"), ...
Information:
Wine: ElyseZinfandel, Region: NapaRegion

Figure 1.2: OMG Four Layer Metadata Architecture

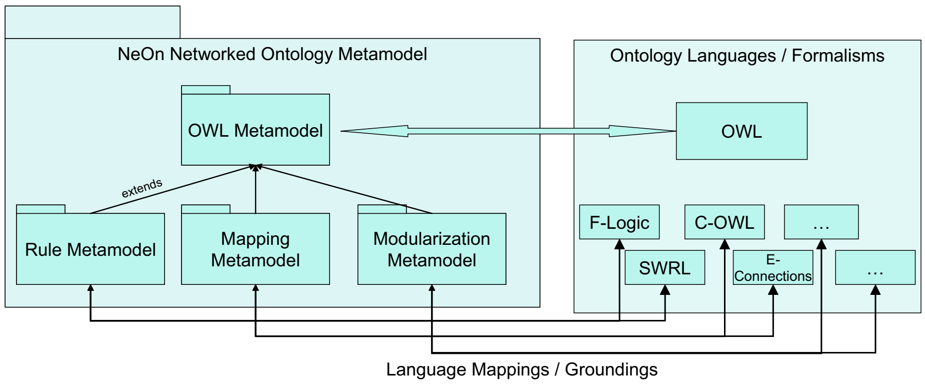

Figure 1.3: Modules of the Ontology Definition Metamodel and possible groundings in ontology languages

the specification of networked ontology model. This visual syntax is based on the metamodel. Mappings in both directions between the metamodel and the profile have to be established.

However, the OWL ODM is just one part of the networked ontology model. Additional modules extend the OWL ODM with specific features of networked ontologies, as shown in Figure 1.3. While the OWL ODM has a direct grounding in the OWL ontology language, the extensions have a generic character in that they are formalism independent and allow a grounding in different formalisms .

1.3 Design Considerations

Modularization The metamodel consists of several modules. The core module, i.e. the OWL metamodel, is extended by different modules that provide additional features, e.g. modularization, mappings, etc. In many application scenarios, only particular aspects of the networked ontology model are needed. In these cases, only the relevant modules need to be supported and used.

Compatibility with standards In terms of the metamodel, two aspects of standards are relevant: The first aspect relates to the fact that we are using a standard formalism - namely the Meta Object Facility - to describe the metamodel. The second aspect relates to the metamodel of networked ontologies itself: A major design goal is compatibility with existing ontology languages. With the Web Ontology Language OWL we have a standard for representing ontologies, therefore we provide a metamodel of OWL directly, with a one-to-one translation. For the other aspects of networked ontologies (mappings, versioning, ...) no such standards exist yet. In favor of general applicability we therefore provide generic metamodels for these extensions that allow translations to different formalisms, as shown in Figure 1.3 .

Chapter 2

The OWL Metamodel

2.1 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[1] 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, an 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 arities 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 [Vol04][Corollary 3.4.1]. Hence, a metamodel for OWL DL necessarily includes OWL Lite.

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

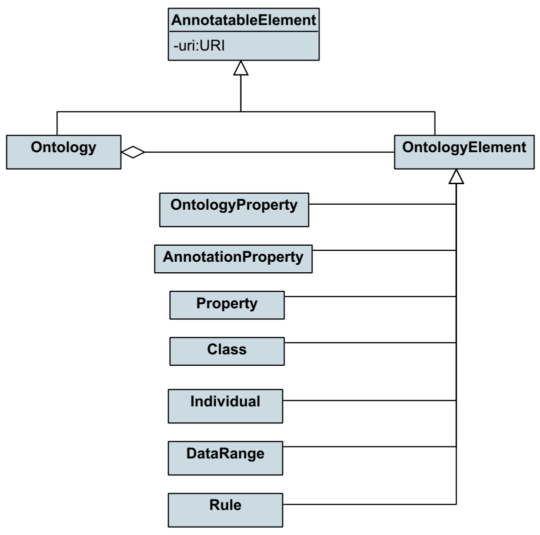

Figure 2.1: Main Elements of the Ontology Definition Metamodel

2.2 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 [Mv03] for a full account of OWL. Our metamodel for OWL DL ontologies ([BVEL04]) has a one-to-one mapping to the abstract syntax of OWL DL and thereby to its formal semantics. It primarily uses basic well-known concepts from UML2. Additionally, the metamodel is augmented with constraints, expressed in the Object Constraint Language ([WK04]), specifying invariants that have to be fulfilled by all models that instantiate the metamodel. The metamodel is augmented with several OCL constraints. Constraints are given in footnotes.

Ontologies

URIs are used to identify all objects in OWL. Figure 2.1 shows the central part of the OWL DL metamodel. Among others, it shows that every element of an ontology is a subclass of the class `OntologyElement` and hence a member of an `Ontology`.

Properties

Properties represent named binary associations in the modeled knowledge domain. OWL distinguishes two kinds of properties, so called object properties and datatype properties. Figure 2.2 shows that both are generalized by the 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

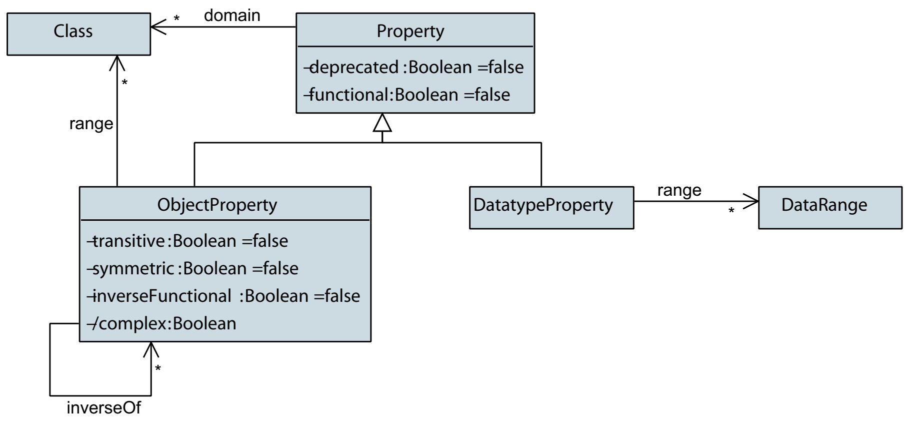
Class
domain
*
Property
-deprecated :Boolean =false
-functional:Boolean =false
*
range
ObjectProperty
-transitive:Boolean =false
-symmetric:Boolean =false
-inverseFunctional :Boolean =false
-/complex:Boolean
*
inverseOf
DatatypeProperty
range
*
DataRange

Figure 2.2: Properties

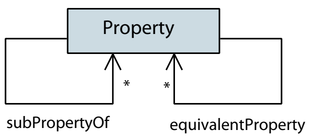

Figure 2.3: Property axioms

may additionally be inverse functional, transitive, symmetric or inverse to another property. Their range is a class[2], while the range of datatype properties is a datarange.

Users can relate properties by using two axioms, modeled as in Figure 2.3. Property subsumption (`subPropertyOf`)[3] 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.

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 repre-

[2]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`.

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

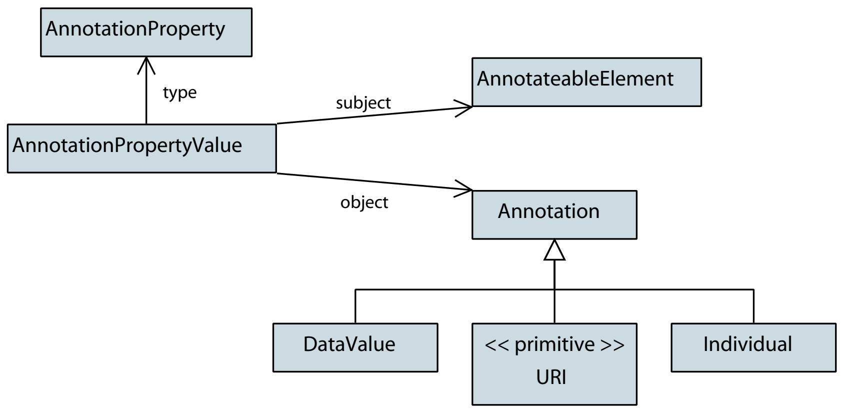

Figure 2.4: Annotations

sented through `OntologyPropertyValue`, which instantiates a certain type of `OntologyProperty` and is a reference between two ontologies.

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` and the object is a Annotation, which can be either a `DataValue`, a `URI` or an `Individual` (cf. Figure 2.4).

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 2.5). 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[4] individuals. Boolean combinations of classes are provided through `Complement`[5], `Intersection` and `Union`.

Restrictions are class constructors that restrict the range of a property for the context of the class (cf. Figure 2.6). Restrictions can be stated *on* datatype and object properties. Accordingly they limit the value *to* a certain datatype or class extension[6]. `UniversalRestriction` provides a form of universal quantification

[4] `oneOf->forAll(name.notEmpty())`

[5] `combinationOf->size()=1`

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

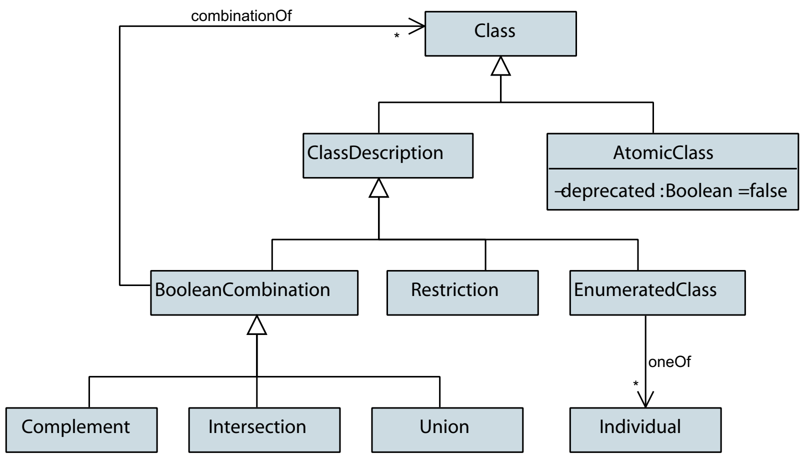

Figure 2.5: Class constructors

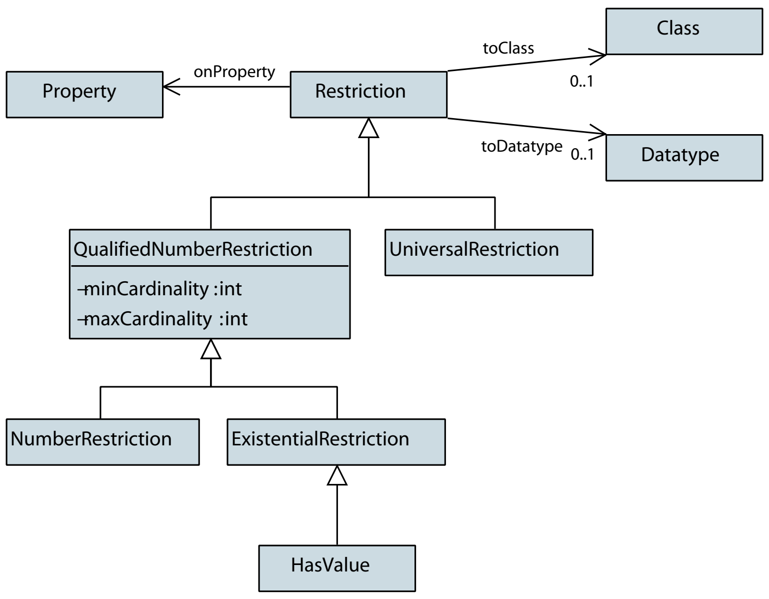

Figure 2.6: OWL Restrictions

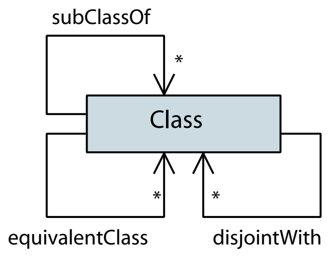

Figure 2.7: Class axioms

that restricts the range of a class to the extension of a certain class or datatype[7].

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[8] while the maximum cardinality should not be smaller than the minimum cardinality[9]. 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[10]. `ExistentialRestricion` can logically and semantically be seen as a special type of qualified number restrictions where the cardinality is fixed[11]. OWL also provides `HasValue`, which is a special type of existential restriction where a given individual is by default in the range of a property[12].

Figure 2.7 shows that classes can be related with each other using class axioms , such as class subsumption (`subClassOf`), class equivalence (`equivalentClass`)[13] and class disjointness (`disjointWith`). These relations between classes are naturally modelled as associations.

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`).

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

[8]`minCardinality>=0`

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

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

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

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

[13]every equivalent class is trivially a superclass: `subClassOf->includesAll(equivalentClass)`

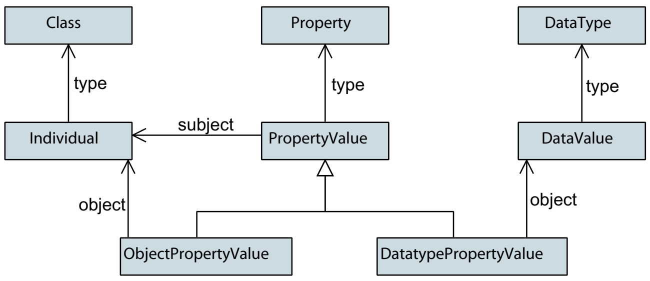

Figure 2.8: Knowledge Base Items

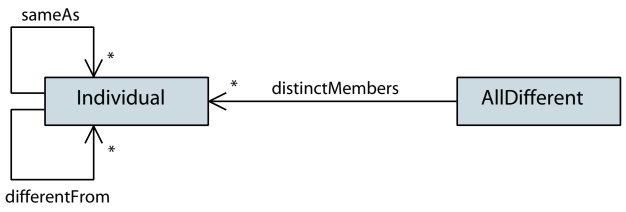

Figure 2.9: Knowledge Base Axioms

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 2.8) are 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, as shown in Figure 2.9. 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[14]. `AllDifferent` is a simpler notation for the pairwise difference of several individuals.

2.3 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

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

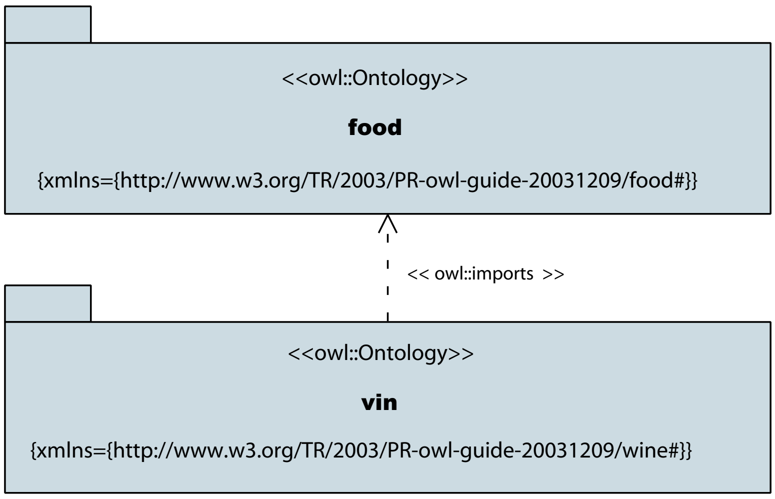
<<owl::Ontology>>
food
{xmlns={http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#}}
<< owl::imports >>
<<owl::Ontology>>
vin
{xmlns={http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#}}

Figure 2.10: owl:imports

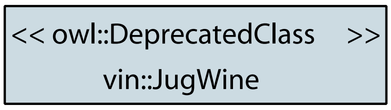

Figure 2.11: owl:DeprecatedClass

ODM can be defined. We heavily rely on the custom stereotypes, which usually carry the name of the corresponding OWL language element.

2.3.1 Ontologies

Figure 2.10 shows that an`Namespace` is represented by a package, 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 2.11, is achieved using a stereotype.

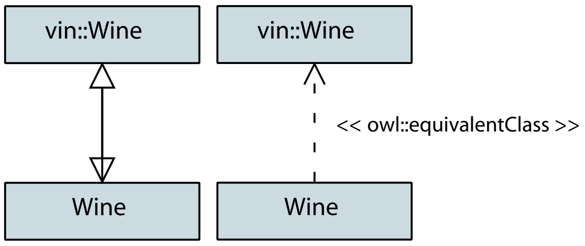

Figure 2.12: owl:EquivalentClass

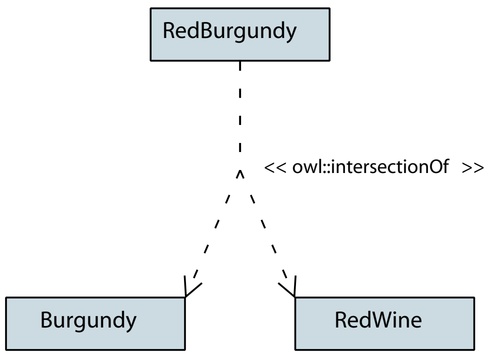

Figure 2.13: owl:intersectionOf

2.3.2 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 2.12. Dependencies could also be used but are not intuitive. Stereotyped UML associations to state class axioms does not translate well to the UML object level. For these reasons, *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 2.13 depicts alternative graphical notations for a intersection of classes. An `EnumeratedClass` is connected to the enumerated individuals by dependencies (cf. Figure 2.14). 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, we depict it as an association to the participating class. Otherwise, in case of a datatype property, it is depicted as an attribute. Figure 2.15 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 cardinal-

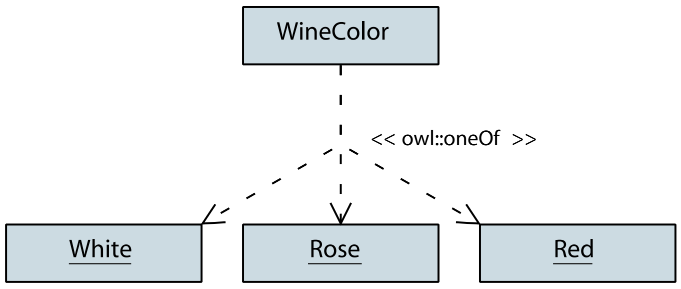

Figure 2.14: owl:oneOf

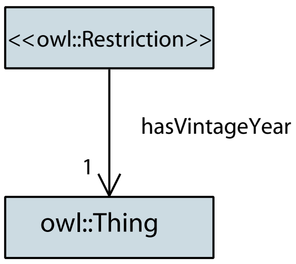

Figure 2.15: owl:cardinality

ity restriction is always owl:Thing and attribute types are rdfs::Literal, which means that they can have every data value.

`ExistentialQuantification` can and `ValueRestriction` have to be indicated by a dedicated stereotype. Figure 2.16 demostrates the notation for an existentially quantified restriction.

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

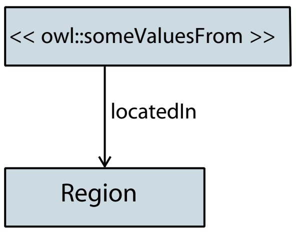

Figure 2.16: owl:someValuesFrom

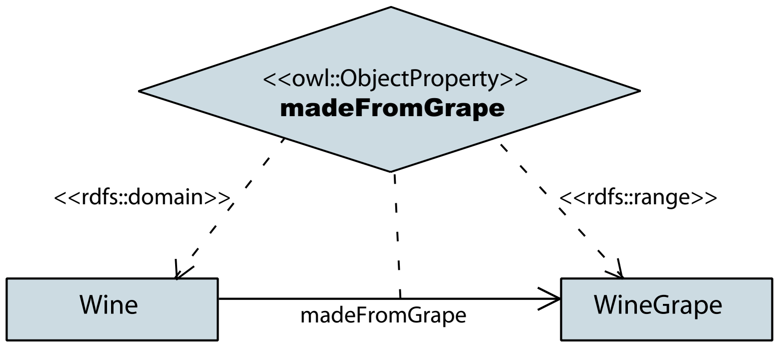

Figure 2.17: An ObjectProperty with domain and range

sounds quite complex, it keeps the consistency with restrictions.

2.3.3 Properties

Object properties are represented as UML n-ary associations[15], 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 2.17).

Analogous to classes, specific properties are assigned a respective stereotype. Figure 2.18 demonstrates the *functionality* and *inverse functionality* stereotype. Naturally, *Deprecation*, *transitivity* and *symmetry*, are represented in the same way. Figure 2.18 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.

2.3.4 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 2.19).

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

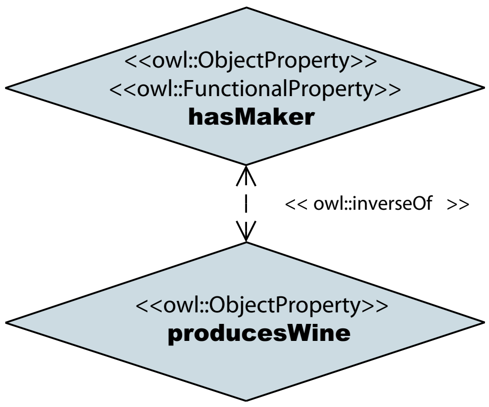

Figure 2.18: Property characteristics

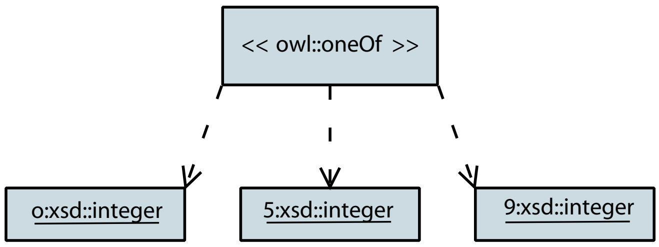

Figure 2.19: Enumerated Datatypes

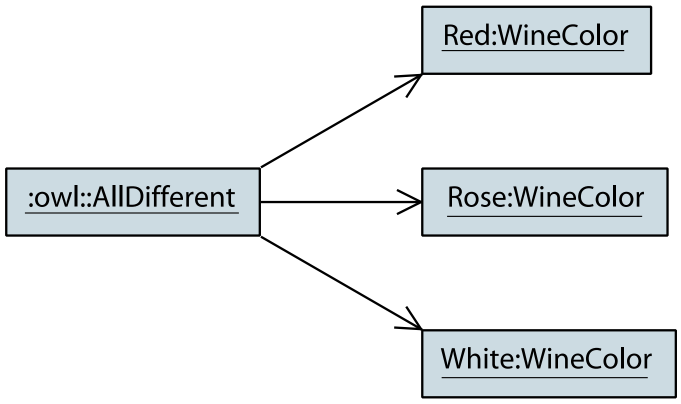

Figure 2.20: owl:AllDifferent

2.3.5 Individuals

Individuals are depicted in the object notation of UML, viz. in the form 'Object : Class'. We represent axioms specifying the equivalence or difference of individuals are represented through stereotyped associations between individuals. We conclude with Figure 2.20, 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.

Chapter 3

The Rule Metamodel

3.1 Design Considerations

3.2 A Metamodel for SWRL Rules

We propose a metamodel for SWRL rules as a consistent extension of the metamodel for OWL DL ontologies which we described in the previous section of this paper. Figure 3.1 shows the metamodel for SWRL rules. We discuss the metamodel step by step along the SWRL specifications. Interested readers may refer to the specifications [HPSB^{+}04] for a full account of SWRL. For a complete reference of the formal correspondence between the metamodel and SWRL itself and the OCL constraints for the rule metamodel, we refer the reader to [BH06b].

3.2.1 Rules

SWRL defines rules as part of an ontology. The SWRL metamodel defines `Rule` as a subclass of `OntologyElement`. `OntologyElement` is defined in the OWL DL metamodel (Figure **??**) as an element of an `Ontology`, via the composition link between `NamedElement` and `Ontology`. As can also be seen in Figure **??**, the class `OntologyElement` is a subclass of the class `AnnotatableElement`, which defines that rules can be annotated. As annotations are modeled in the ODM, a URI reference can be assigned to a rule for identification.

A rule consists of an antecedent and a consequent, also referred to as body and head of the rule, respectively. Both the antecedent and the consequent consist of a set of atoms which can possibly be empty, as depicted by the multiplicity in Figure 3.1. Informally, a rule says that *if* all atoms of the antecedent hold, *then* the consequent holds. An empty antecedent is treated as trivially true, whereas an empty consequent is treated as trivially false.

The same antecedent or consequent can be used in several rules, as indicated in the metamodel by the multiplicity on the association between `Rule` on the one hand and Antecedent or Consequent on the other hand. Similarly, the multiplicities of the association between Antecedent and Atom and of the association between Consequent and Atom define that an antecedent and a consequent can hold zero or more atoms. The multiplicity in the other direction defines that the same atom can appear in several antecedents or consequents. According to the SWRL specifications, every Variable that occurs in the Consequent of a rule must occur in the Antecedent of that rule, a condition referred to as "safety".

3.2.2 Atoms, Terms and Predicate symbols

The atoms of the antecedent and the consequent consist of predicate symbols and terms. According to SWRL, they can have different forms:

- $C(x)$, where C is an OWL description and x an individual variable or an OWL individual, or C is an OWL data range and x either a data variable or an OWL data value;
- $P(x, y)$, where P is an OWL individual valued property and x and y are both either an individual variable or an OWL individual, or P is an OWL data valued property, x is either an individual variable or an OWL individual and y is either a data variable or an OWL data value;
- sameAs(x, y), where x and y are both either an OWL individual or an individual variable;
- differentFrom(x, y), where x and y are both either an OWL individual or an individual variable;
- builtIn$(r, x, ...)$, where r is a built-in predicate and x is a data variable or OWL data value. A builtIn atom could possibly have more than one variable or OWL data value.

The first of these, OWL description, data range and property, were already provided in the ODM, namely as metaclasses Class, DataRange and Property, respectively. As can be seen in Figure 3.1, the predicates Class, DataRange, Property and BuiltIn are all defined as subclasses of the class PredicateSymbol, which is associated to Atom. The remaining two atom types, sameAs and differentFrom, are represented as specific instances of PredicateSymbol.

To define the order of the atom terms, we put a class TermOrder in between Atom and Term. This UML association class connects atoms with terms and defines the term order via the attribute order.

3.3 A UML Profile for Rules

UML provides an extension mechanism, the UML profile mechanism, to tailor the language to specific application areas. The definition of such a UML extension is based on the standard UML metamodel. In this section, we propose a

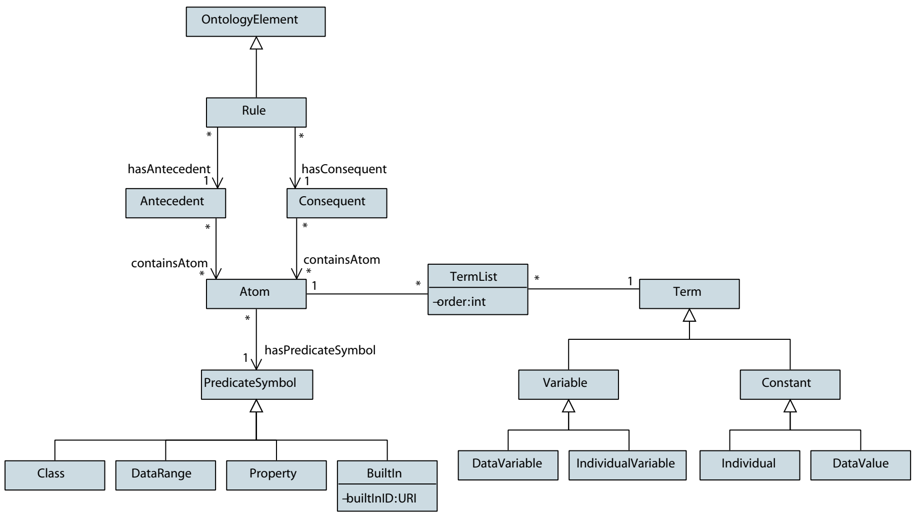

Figure 3.1: The Rule Definition Metamodel

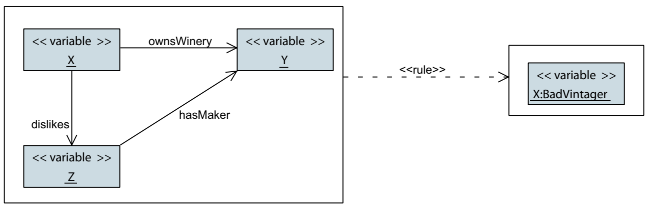

Figure 3.2: BadVintager(x) $\leftarrow$ ownsWinery(x, y) $\wedge$ dislikesWine(x, z) $\wedge$ hasMaker(z, y)

UML profile for modeling SWRL rules which is consistent with the design considerations taken for the basic UML Ontology Profile. For a complete reference of the relationship between the UML profile and the metamodel introduced in Section 3.2, we refer the reader to [BH06b]. Figure 3.2 shows an example of a rule, which defines that when a vintager does not like the wine made in his winery, he is a bad vintager. We introduce the profile in an order based on the SWRL metamodel introduced in Section 3.2.

3.3.1 Rules

As can be seen in Figure 3.2, a rule is depicted by two boxes connected via a dependency with the stereotype `rule`. All atoms of the antecedent are contained in the box at the origin of the dependency, whereas the box at the end

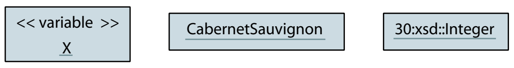

Figure 3.3: Terms

contains the consequent. This way, antecedent and consequent can easily be distinguished, and it also allows to distinguish between the rule atoms and the OWL DL facts which are depicted in similar ways. The left box of our example contains the three variable definitions and the three properties that are defined between these variables. The consequent-box on the right contains the definition of the variable X from which it is known which class it belongs to. We explain the specific design considerations of these concepts in the following subsections.

3.3.2 Terms

Although the existing UOP already comprises a visual syntax for individuals and data values, namely by applying the UML object notation, it does not include a notation for variables since OWL DL ontologies do not contain variables. We decided to depict variables in the UML object notation as well, since a variable can be seen as a partially unknown class instance. We provide a stereotype `variable` to distinguish a variable. Figure 3.3 shows a simple example for a variable, an individual and a data value.

3.3.3 Predicate Symbols in Atoms

Class description and data range.

A visual notation for individuals as instances of class descriptions is already provided in the UOP for OWL DL. An atom with a class description and a variable as its term, is illustrated similarly. An appropriate stereotype is added. An example of this can be seen in the consequent in Figure 3.2. A visual construct for a data range definition using individuals is contained in the UOP for OWL DL as well, namely represented in the same way as class individuals. Data range constructs containing variables are also depicted in a similar fashion.

Properties.

Object properties are depicted as directed associations between the two involved elements. A datatype property is pictured as an attribute. These notations were provided for properties of individuals by the UOP for OWL DL, and we follow them to depict properties of variables. The antecedent of the rule in Figure 3.2 contains three such object properties between variables, `ownsWinery`,

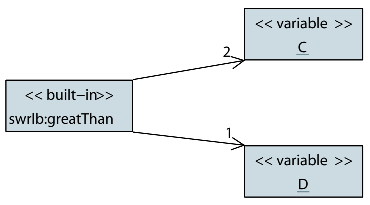

Figure 3.4: Built-in predicates

`dislikesWine` and `hasMaker`. The other example rule, depicted in Figure 3.5, contains amongst other things twice the datavalued property `yearValue`.

sameAs and differentFrom.

According to the UOP, equality and inequality between objects are depicted using object relations. Because of the similarity between individuals and variables, as shortly explained in Section 3.3.2, we propose to use the same visual notation for `sameAs` and `differentFrom` relations between two variables or between a variable and an object.

Built-in predicates.

For the visual representation of built-in relations, we use usual associations to all participating variables and data values, similar to the `owl:AllDifferent` concept provided in the basic UOP. To denote the built-in relation, we provide the stereotype `built-in` together with the specific built-in ID. The names of the associations denote the order of the arguments, by numbers. Figure 3.4 shows an example of a built-in relation `swrlb:greaterThan`, which is defined to check whether the first involved argument is greater than the second one. For the six most basic built-ins, `swrlb:equal`, `swrlb:notEqual`, `swrlb:lessThan`, `swrlb:lessThanOrEqual`, `swrlb:greaterThan` and `swrlb:greaterThanOrEqual`, we provide an alternative notation. Instead of depicting the stereotype and the name of the built-in, an appropriate icon can be used. Figure 3.5 depicts a rule example using this alternative notation for built-in predicates. This rule states that if the year value of a wine (y) is greater than the year value of another wine (x), then the second wine (x) is older than the first one (y). Next to the built-in predicate, Figure 3.5 shows six variables with the properties `hasVintageYear`, `yearValue` and `olderThan`.

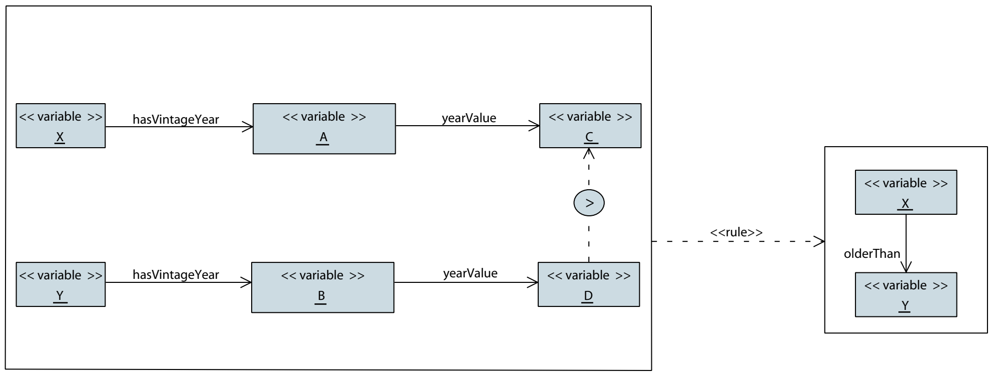

Figure 3.5: olderThan$(x, y) \leftarrow$ hasVintageYear$(x, u) \wedge$ hasVintageYear$(y, v) \wedge$ yearValue$(u, w) \wedge$ yearValue$(v, z) \wedge$ swrlb:greaterThan(z, w)

Chapter 4

The Mapping Metamodel

4.1 Ontology Mapping Formalisms

In contrast to the area of ontology languages where the Web Ontology Language OWL has become a de facto standard for representing and using ontologies, there is no agreement yet on the nature and the right formalism for defining mappings between ontologies. In a recent discussion on the nature of ontology mappings, some general aspects of mapping approaches have been identified [SU05]. We briefly discuss these aspects in the following and clarify our view on mappings that is reflected in the proposed metamodel with respect to these aspects.

4.1.1 What do mappings define ?

In this paper, we restrict our attention to declarative mapping specifications. In particular, we see mappings as axioms that define a semantic relation between elements in different ontologies.

A number of different kinds of semantic relations have been proposed. Most common are the following kinds of semantic relations:

Equivalence ($\equiv$) Equivalence states that the connected elements represent the same aspect of the real world according to some equivalence criteria. A very strong form of equivalence is equality, if the connected elements represent exactly the same real world object.

Containment ($\sqsubseteq, \sqsupseteq$) Containment states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. Depending on which of the elements is more specific, the containment relation is defined in the one or in the other direction.

Overlap (o) Overlap states that the connected elements represent different aspects of the world, but have an overlap in some respect. In particular, it states that some objects described by the element in the one ontology may also be described by the connected element in the other ontology.

In some approaches, these basic relations are supplemented by their negative counterparts. The corresponding relations can be used to describe that two elements are *not* equivalent ($\not\equiv$), *not* contained in each other ($\not\sqsubseteq$) or *not* overlapping or disjoint respectively ($\emptyset$). Adding these negative versions of the relations leaves us with eight semantic relations that cover all existing proposals for mapping languages. In addition to the type of semantic relation, an important distinction is whether the mappings are to be interpreted as extensional or as intensional relationships.

Extensional In extensional mapping definitions, the semantic relations are interpreted as set-relations between the sets of objects represented by elements in the ontologies. Intuitively, elements that are extensionally the same have to represent the same set of objects.

Intensional In the case of intensional mappings, the semantic relations relate the elements directly, i.e. considering the properties of the element itself. In particular, if two elements are intensionally the same, they refer to exactly the same real world object.

4.1.2 What do mappings preserve ?

It is normally assumed that mappings preserve the 'meaning' of the two models in the sense that the semantic relation between the intended interpretations of connected elements is the one specified in the mapping. A problem with this assumption is that it is virtually impossible to verify this property. Instead, there are a number of verifiable formal properties that mappings can be required to satisfy. Examples of such formal properties are the satisfiability of the overall model, the preservation of possible inferences or the preservation of answers to queries. Often, such properties can only be stated relative to a given application context, such as a set of queries to be answered or a set of tasks to be solved.

The question of what is preserved by a mapping is tightly connected to the hidden assumptions made by different mapping formalisms. A number of important assumptions that influence this aspect have been identified and formalized in [SSW05]. A first basic distinction concerns the relationship between the sets of objects (domains) described be the mapped ontologies. Generally, we can distinguish between a global domain and local domain assumption:

Global Domain It is assumed that both ontologies describe exactly the same set of objects. As a result, semantic relations are interpreted in the same way as axioms in the ontologies. There are special cases of this assumption, where one ontology is regarded as a 'global schema' and describes the set of all objects, other ontologies are assumed to describe subsets of these objects.

Local Domains It is not assumed that ontologies describe the same set of objects. This means that mappings and ontology axioms normally have different semantics. There are variations of this assumption in the sense that sometimes it is assumed that the sets of objects are completely disjoint and sometimes they are assumed to overlap each other.

These assumptions about the relationship between the domains is especially important for extensional mapping definitions, because in cases where two ontologies do not talk about the same set of instances, the extensional interpretation of a mapping is problematic as classes that are meant to represent the same aspect of the world can have disjoint extensions. In such cases, e.g. in C-OWL [BGvH+03], the relationship is not defined directly as a set relationship between the extensions of the concepts, but can be defined in terms of domain relations that connect the interpretation domains by codifying how elements in one domain map into elements of the other domain.

Other assumptions made by approaches concerns the use of unique names for objects - this assumption is often made in the area of database integration - and the preservation of inconsistencies across mapped ontologies. In order to make an informed choice about which formalism to use, these assumptions have to be represented by the modeler and therefore need to be part of the proposed metamodel.

4.1.3 What do mappings connect ?

In the context of this work, we decided to focus on mappings between ontologies represented in OWL DL. This restriction makes it much easier to deal with this aspect of ontology mappings as we can refer to the corresponding metamodel for OWL DL specified in [BH06a]. In particular, the metamodel contains the class `OntologyElement`, that represents an arbitrary part of an ontology specification. While this already covers many of the existing mapping approaches, there are a number of proposals for mapping languages that rely on the idea of view-based mappings and use semantic relations between (conjunctive) queries to connect models, which leads to a considerably increased expressiveness.

4.1.4 How are mappings organized ?

The final question is how mappings are organized. They can either be part of a given model or be specified independently. In the latter case, the question is how to distinguish between mappings and other elements in the models. Mappings can be uni- or bidirectional. Further, it has to be defined whether a set of mappings is normative or whether it is possible to have different sets of mappings according to different applications, viewpoints or different matchers.

In this work, we use a mapping architecture that has the greatest level of generality in the sense that other architectures can be simulated. In particular, we make the following choices:

- A mapping is a set of mapping assertions that consist of a semantic relation between mappable elements in different ontologies

- Mappings are first-class objects that exist independent of the ontologies. Mappings are directed and there can be more than one mapping between two ontologies

These choices leave us with a lot of freedom for defining and using mappings. Approaches that see mappings as parts of an ontology can be represented by the ontology and a single mapping. If only one mapping is defined between two ontologies, this can be seen as normative, and bi-directional mappings can be described in terms of two directed mappings.

4.2 A Metamodel for Ontology Mappings

We propose a formalism-independent metamodel for OWL ontology mappings. The metamodel is a consistent extension the metamodels for OWL DL ontologies and rules. It has constraints defined in OCL [WK04] as well, which are specified in footnotes.

Figure 4.1 shows the metamodel for mappings. In the figures, darker grey classes denote classes from the metamodels of OWL DL and rule extensions.

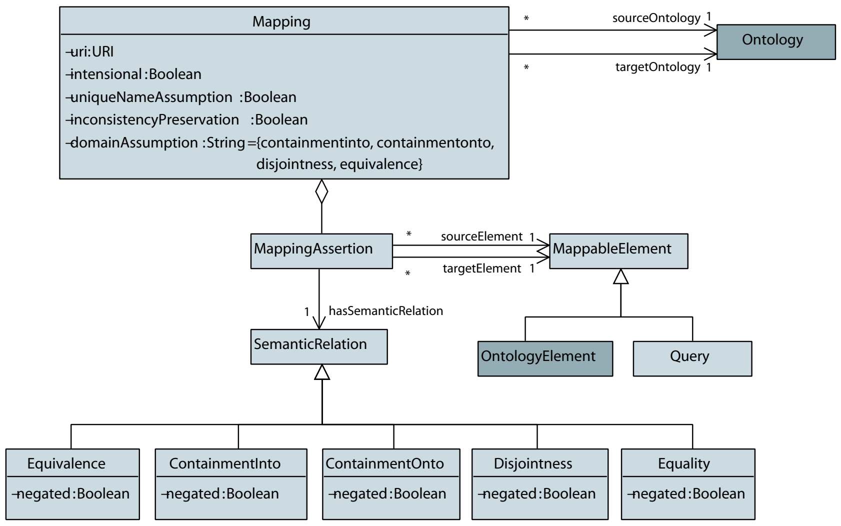

Figure 4.1: Metamodel for ontology mappings

The central class in the metamodel is the class `Mapping`, having four attributes. The URI, defined by the attribute `uri`, allows to uniquely identify a mapping and refer to it as a first-class object. The assumptions about the use of unique names for objects and the preservation of inconsistencies across mapped ontologies, are defined through the boolean attributes `uniqueNameAssumption` respectively `inconsistencyPreservation`. For the assumptions about the domain, we defined an attribute `DomainAssumption`. This attribute may take

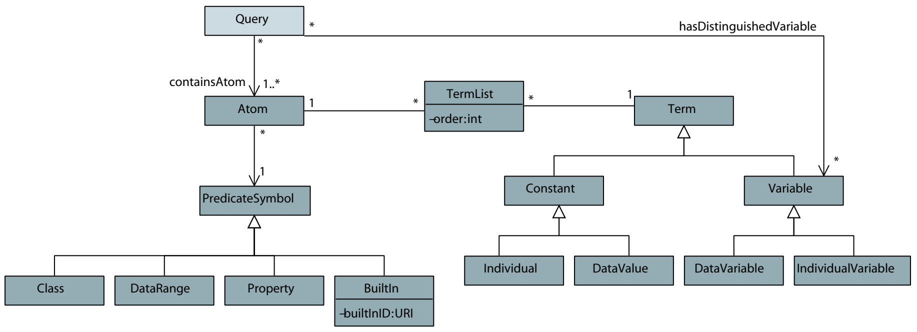

Figure 4.2: Metamodel for ontology mappings - definition of a query

specific values that describe the relationship between the connected domains: overlap, containment (in one of the two directions) or equivalence.

A mapping is always defined between two ontologies. An ontology is represented by the class `Ontology` in the OWL DL metamodel. Two associations from `Mapping` to `Ontology`, `sourceOntology` and `targetOntology`, specify the source respectively the target ontology of the mapping. Cardinalities on both associations denote that to each `Mapping` instantiation, there is exactly one `Ontology` connected as source and one as target.

A mapping consists of a set of mapping assertions, denoted by the MOF aggregation relationship between the two classes `Mapping` and `MappingAssertion`. The elements that are mapped in a `MappingAssertion` are defined by the class `MappableElement`. A `MappingAssertion` is defined through exactly one `SemanticRelation`, one source `MappableElement` and one target `MappableElement`. This is defined through the three associations starting from `MappingAssertion` and their cardinalities.

In Section 4.1.1, we have introduced four semantic relations along with their logical negation to be defined in the metamodel. Two of these relationship types are directly contained in the metamodel through the subclasses `Equivalence` and `Overlap` of the class `SemanticRelation`. The other two, containment in either direction, are defined through the subclass `Containment` and its additional attribute `direction`, which can be `sound` ($\sqsubseteq$) or `complete` ($\sqsupseteq$).

The negated versions of all semantic relations are specified through the boolean attribute `negated` of the class `SemanticRelation`. The other attribute of `SemanticRelation`, `interpretation`, defines whether the mapping assertion is assumed to be interpreted intentionally or extensionally.

As discussed in Section 4.1, a mapping assertion can connect two mappable elements, which may ontology elements or queries. To support this, `MappableElement` has two subclasses `OntologyElement` and `Query`. The former is previously defined in the OWL DL metamodel, as shown in Figure **??**. The class `Query` reuses constructs from the SWRL metamodel. The reason for reusing large parts of the rule metamodel lies in the fact that conceptually rules and queries are of very similar nature [TF05]: A rule consists of a rule body

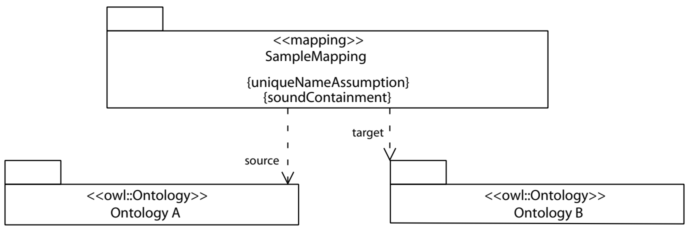

Figure 4.3: Visual notation for a mapping between two ontologies

(antecedent) and rule head (consequent), both of which are conjunctions of logical atoms. A query can be considered as a special kind of rule with an empty head. The distinguished variables specify the variables that are returned by the query. Informally, the answer to a query consists of all variable bindings for which the grounded rule body is logically implied by the ontology. Figure 4.2 shows this connection and shows how a `Query` is composed. They depict how atoms from the antecedent and the consequent of SWRL rules can be composed. Similarly, a `Query` also contains a `PredicateSymbol` and some, possibly just one, `Terms`. We defined the permitted predicate symbols through the subclasses `Class`, `DataRange`, `Property` and `BuiltIn`. Similarly, the four different types of terms are specified as well. The UML association class `TermList` between `Atom` and `Term` allows to identify the order of the atom terms. Distinguished variables of a query are differentiated through an association between `Query` and `Variable`.

4.3 A UML Profile for Ontology Mappings

This section describes the UML profile as a visual notation for specifying ontology mappings, based on the metamodel discussed in Section 4.2. Our goal is to allow the user to specify mappings without having decided yet on a specific mapping language or even on a specific semantic relation. This is reflected in the proposed visual syntax which is, like the metamodel, independent from a concrete mapping formalism. The UML profile is consistent with the design considerations taken for the previously defined UML profiles for OWL ontologies and rule extensions.

First of all, users specify two ontologies between which they want to define mappings. The visual notation for this as defined in our profile, is presented in Figure 4.3. Just as for ontologies as collections of ontology elements, we apply the UML grouping construct of a package to represent mappings as collections of mapping assertions. Attributes of the mapping, like the domain assumption, are represented between curly brackets.

In Figure 4.4, a source concept `Publication` is defined to be a special case of the target concept `Entry`. Figure 4.5 models `Researcher Fowler` and `Author MartinFowler` as two equivalent instances. The third example in Figure 4.6

relates two properties `authorOf` and `creatorOf` using an extensional containment relationship. Both source and target elements of mapping assertions

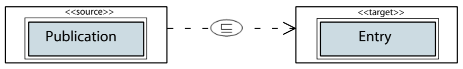

Figure 4.4: Sample containment relation between two concepts

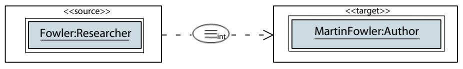

Figure 4.5: Sample intensional equivalence relation between two individuals

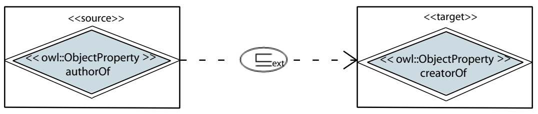

Figure 4.6: Sample extensional equivalence relation between two properties

are represented in a box, connected to each other via a dependency with the corresponding symbol of the semantic relation. In the first step of the process, when users just mark elements being semantically related without specifying the type of semantic relation, the dependency does not carry any relation symbol. Stereotypes in the two boxes denote source- and target ontology. Like defined in the metamodel, these mapped elements can be any element of an ontology (metaclass `OntologyElement`) or a query (metaclass `Query`). They are represented like defined in the UML profile for OWL and rules. The parts of the mappable elements which are effectively being mapped to each other, are denoted via a double-lined box, which becomes relevant if the mapped elements are more complex constructs, as explained in the following.

A more complex example mapping assertion is pictured in Figure 4.7. The example defines that the union of the classes `PhDThesis` and `MasterThesis`, is equivalent to the class `Thesis`.

Figure 4.8 shows another example of an equivalence relation between two expressions. It specifies that the class which is connected to the class `Publication` via a property `authorOf` with the `someValuesFrom` restriction, is equivalent to the class `Author`.

Figure 4.9 shows an example of an equivalence relation between two queries. The first query is about a Publication X with a Topic Y named Z. The target query is about an Entry X with subject Z. The mapping assertion defines the two queries to be equivalent. The effective correspondences are established between the the two distinguished variables X and Z, which are again denoted with a double-lined box.

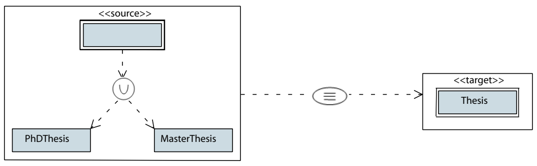

Figure 4.7: Sample equivalence relation between complex class descriptions

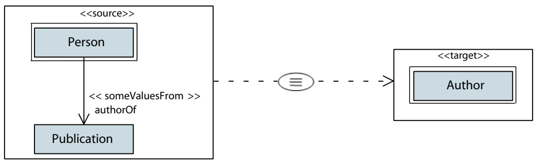

Figure 4.8: Sample equivalence relation between complex class descriptions

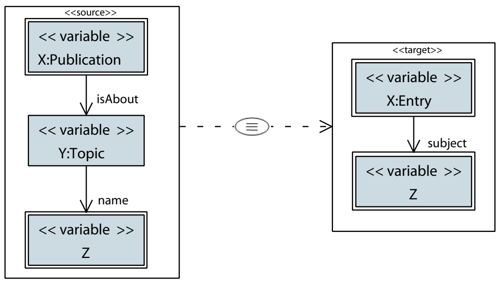

Figure 4.9: Sample equivalence relation between two queries

Bibliography

[BGvH+03] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *Second International Semantic Web Conference ISWC'03*, volume 2870 of *LNCS*, pages 164–179. Springer, 2003.

[BH06a] Saartje Brockmans and Peter Haase. A Metamodel and UML Profile for Networked Ontologies – A Complete Reference. Technical report, Universität Karlsruhe, April 2006. `http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/ontology-metamodeling.pdf`.

[BH06b] Saartje Brockmans and Peter Haase. A Metamodel and UML Profile for Rule-extended OWL DL Ontologies –A Complete Reference. Technical report, Universität Karlsruhe, March 2006. `http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/owl-metamodeling.pdf`.

[BVEL04] Saartje Brockmans, Raphael Volz, Andreas Eberhart, and Peter Loeffler. Visual modeling of owl dl ontologies using uml. In *Proceedings of the Third International Semantic Web Conference*, pages 198–213, Hiroshima, Japan, November 2004. Springer.

[HEC+04] Lewis Hart, Patrick Emery, Bob Colomb, Kerry Raymond, Sarah Taraporewalla, Dan Chang, Yiming Ye, and Mark Dutra Elisa Kendall. OWL full and UML 2.0 compared, March 2004. `http://www.itee.uq.edu.au/$\sim$colomb/Papers/UML-OWLont04.03.01.pdf`.

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. World Wide Web Consortium, May 2004. W3C Member Submission, `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`.

[IBM05] IBM, Sandpiper Software. *Ontology Definition Metamodel, Fourth Revised Submission to OMG*, November 2005.

[Mv03] 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/.

[Obj02] Object Management Group. Meta Object Facility (MOF) Specification. Technical report, Object Management Group (OMG), April 2002. `http://www.omg.org/docs/formal/02-04-03.pdf`.

[SSW05] L. Serafini, H. Stuckenschmidt, and H. Wache. A formal investigation of mapping languages for terminological knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI05*, Edinburgh, UK, August 2005.

[SU05] Heiner Stuckenschmidt and Michael Uschold. Representation of semantic mappings. In Yannis Kalfoglou, Marco Schorlemmer, Amit Sheth, Steffen Staab, and Michael Uschold, editors, *Semantic Interoperability and Integration. Dagstuhl Seminar Proceedings*, volume 04391, Germany, 2005. IBFI, Schloss Dagstuhl.

[TF05] Sergio Tessaris and Enrico Franconi. Rules and queries with ontologies: a unifying logical framework. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Description Logics*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[Vol04] 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.

[WK04] Jos Warmer and Anneke Kleppe. *Object Constraint Language 2.0*. MITP Verlag, 2004.