A Model Driven Approach for Building OWL DL and OWL Full Ontologies

Saartje Brockmans[1], Robert M. Colomb[2], Peter Haase[1], Elisa F. Kendall[3], Evan K. Wallace[4], Chris Welty[5], and Guo Tong Xie[6]

[1] AIFB, Universität Karlsruhe (TH), Germany
[2] School of Information Technology and Electrical Engineering, The University of Queensland, Australia
[3] Sandpiper Software, Inc., Los Altos, California
[4] US National Institute of Standards and Technology, Gaithersburg, Maryland
[5] IBM Watson Research Center, New York
[6] IBM China Research Lab, China

Abstract. This paper presents an approach for visually modeling OWL DL and OWL Full ontologies based on the well-established visual modeling language UML. We discuss a metamodel for OWL based on the Meta-Object Facility, an associated UML profile as visual syntax, and transformations between both. The work we present supports model-driven development of OWL ontologies and is currently undergoing the standardization process of the Object Management Group. After describing our approach, we present the implementation of our approach and an example, showing how the metamodel and UML profile can be used to improve developing Semantic Web applications.

1 Introduction

The standardization of the Web Ontology Language (OWL, [8]) by the World Wide Web Consortium (W3C) contributed heavily to the wide-spread use of ontologies. In 2003, the Object Management Group (OMG), a standardization consortium for various aspects of software engineering including the well-established Unified Modeling Language (UML, [24]), replied to this by issuing a Request for Proposal for an Ontology Definition Metamodel (ODM, [18]). The intention was to provide a Meta-Object Facility (MOF, [23]) based metamodel to support the development of ontologies using UML modeling tools and the two-way transformation between ontologies written in a specific ontology representation language and ontologies modeled using a dedicated UML syntax. Since that time, a submission team has developed a submission (see [7] for a concise overview) which has undergone several revisions, based on comments solicited not only of the OMG but from the W3C, ISO and Semantic Web communities as well.

The ODM submission supports the knowledge representation languages OWL [8], RDF [1], Common Logic [15] and Topic Maps [14]. The modular structure of MOF makes it straightforward for third parties to extend and enhance the metamodel.

I. Cruz et al. (Eds.): ISWC 2006, LNCS 4273, pp. 187–200, 2006.

This paper focuses on the OWL portions of the ODM submission, which is currently in adoption recommendation vote at OMG. It supports model-driven development of OWL DL as well as OWL Full ontologies using UML and two-way transformations between ontologies modeled in OWL and ontologies modeled using the UML profile. We have not explicitly covered OWL Lite, but all constructs are provided in the base OWL and OWL DL packages. The paper starts with an introduction of the Model Driven Architecture and its Meta-Object Facility, and UML profiles in Section 2. Then, the metamodel for OWL, the associated UML profile and the transformations between the different models are described in Section 3. Section 4 shows the implementation of our approach and an example. Finally, after discussing related work in Section 5, we conclude by summarizing our work and addressing future investigations in Section 6.

2 Background

2.1 Model Driven Architecture and the Meta-object Facility

Before presenting the model-driven approach to ontology engineering in the next sections, we summarize the Object Management Group's Model Driven Architecture (MDA, [5]) and its Meta-Object Facility (MOF, [23]), which is one of the main pillars of our approach.

In the history of software engineering, there has been a notable increase of the use of models and the level of abstraction in the models. The basic idea of MDA is that the system functionality is defined as a platform-independent model, using an appropriate specification language and then translated to one or more platform-specific models for the actual implementation. To accomplish this goal, MDA defines an architecture that provides a set of guidelines for structuring specifications expressed as models. The translation between a platform-independent model and platform-specific models is often performed using automated tools.

MDA comprises of a four-layer metamodel architecture: meta-metamodel (M3) layer, metamodel (M2) layer, model (M1) layer, and instance (M0) layer. At the top of the MDA architecture is the meta-metamodel, i.e., MOF. It defines an abstract language and framework for specifying, constructing and managing technology neutral metamodels. It is the foundation for defining any modeling language such as UML. MOF also defines a framework for implementing repositories that hold metadata (models) described by metamodels. The main objective of having the four layers with a common meta-metamodel is to support multiple metamodels and models and to enable their extensibility, integration and generic model and metamodel management. Note that the meta-metamodel layer is hard wired in the sense that it is fixed, while the layer of the metamodels is flexible and allows expression of various metamodels. All metamodels, standard or custom, defined by MOF are positioned at the M2 layer. One of these is UML, a graphical modeling language for specifying, visualizing and documenting software systems. The models of the real world, represented by concepts defined in the corresponding metamodel at M2 layer (e.g., UML metamodel) are at M1

layer. Finally, at M0 layer, are objects from the real world or information objects representing these in an information system.

A MOF-based metamodel has clear advantages being based on a standard meta-metamodelling system with a well-developed suite of software tools and integrated transformation possibilities with other MOF-based metamodels.[11].

2.2 UML Profiles

UML methodology, tools and technology seem to be a feasible approach for supporting the development and maintenance of ontologies. The UML class diagram is a rich representation system, widely used, and well-supported with software tools. However, an ontology cannot be sufficiently represented in UML [12] and a dedicated visual ontology modeling language is needed. The two representations share a set of core functionalities 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.

The UML profile mechanism is an extension mechanism to tailor UML to specific application areas. UML profiles provide specializations, using stereotypes, of existing UML constructs. They are grounded in MOF, in that they are defined in terms of the MOF meta-metamodel. Moreover, they are based on the UML Kernel package and the Profiles section defined in [21].

3 Approach

In this section, we present a MOF-based metamodel for OWL DL and OWL Full. Models based on these metamodels are OWL ontologies. OWL constructs have a direct correspondence with those of the metamodel. Analogously, we define a MOF-based UML profile, which is instantiated by concrete UML models, to enable the use of UML notation and tools for ontology modeling. Within the MOF framework, the UML models are transformed into OWL definitions and vice versa.

3.1 A Metamodel for OWL DL and OWL Full

Overview and Design Considerations. As mentioned in Section 1, although we focus on OWL in this paper, the ODM submission at OMG provides metamodels for several knowledge representation languages. All these are independent of each other, except the OWL metamodel which extends the RDFS metamodel, as the OWL language itself extends the RDF-S language. The metamodel for OWL specifically, contains three packages. First of all, the primary OWLBase package contains the metamodel constructs common to both OWL DL and OWL Full. Two additional subpackages, the OWLDL package and the OWLFull package, contain constraints and extensions required to distinguish the two dialects OWL DL and OWL Full from one another, as explained in more detail later in

this section. Users can elect to support the primary package and either or both of the subordinate packages in order to have complete coverage of either or both dialects of OWL. All metamodel packages are provided with constraints in the Object Constraint Language (OCL, [20]). These expressions specify invariant conditions that must hold for the ontologies being modeled. For the constraints on the metamodel, we refer the user to [13].

We now go through the different parts of the OWLBase metamodel package and show some of the diagrams. Subsequently, we introduce the OWLDL and OWLFull packages.

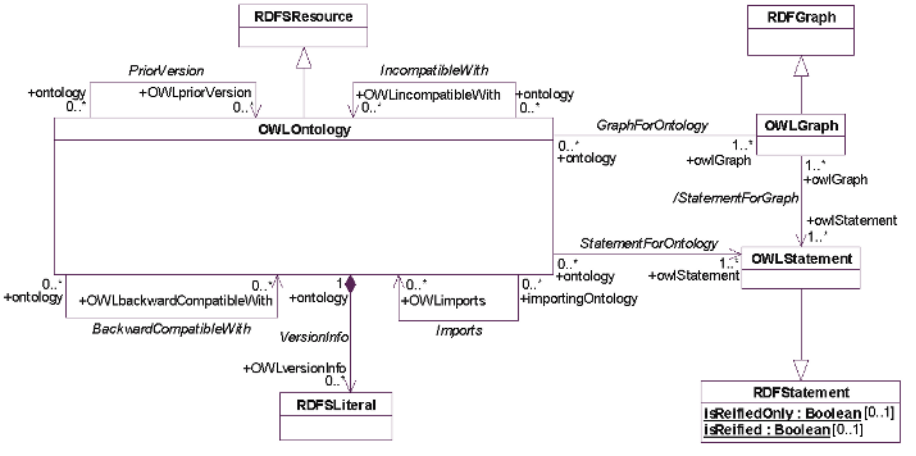

Fig. 1. The Ontology Diagram

OWLBase Package - OWL Ontology. The RDF metamodel represents an `RDFStatement` as a triple, containing subject, predicate and object whereas an `RDFGraph` is a set of triples (`RDFStatements`). As shown in Figure 1, the `OWLGraph` class specifies the subset of RDF graphs that are valid OWL graphs, consisting of all OWL expressions. Similarly, the subset of RDF statements that are valid OWL statements is reflected by the `OWLStatement` class. The distinction between `OWLStatement` and `RDFStatement` is required, as in OWL DL not every `RDFStatement` is a valid `OWLStatement`. An ontology is identified by a URI reference (inherited from `RDFSResource`), which allows us to make statements about that ontology.

OWLBase Package - Class Descriptions. The metamodel has a class `OWLClass` for simple OWL class definitions defined as a special type of `RDFSClass`. Moreover, it has subclasses which represent special types of OWL class descriptions: `ComplementClass`, `EnumeratedClass`, `IntersectionClass`,

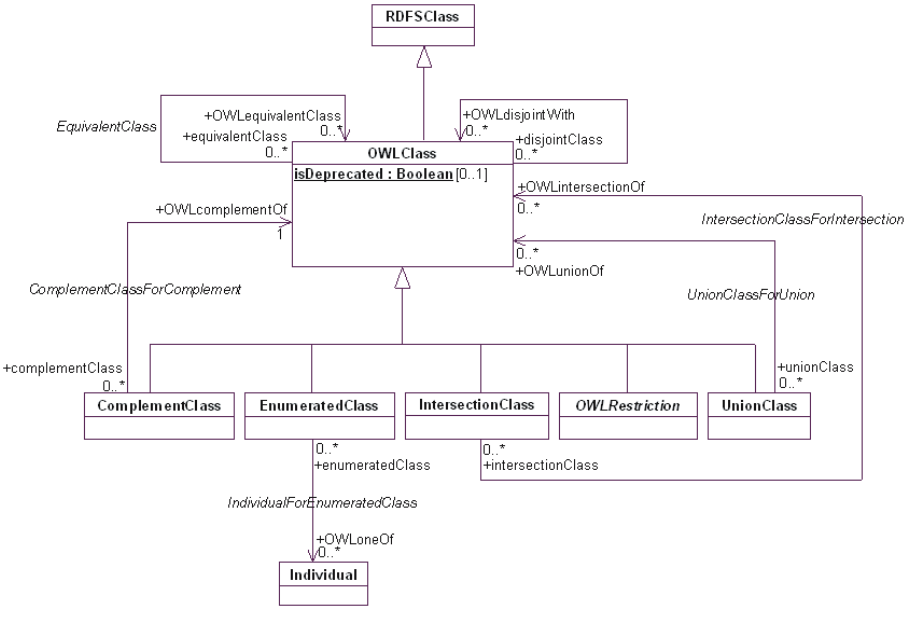

Fig. 2. OWL Class Descriptions

`OWLRestriction` and `UnionClass`. An `EnumeratedClass` is connected to `Individuals` through an association role `OWLoneOf`. Associatons between the classes define the classes in the class descriptions, e.g. the association `IntersectionClassForIntersection` between `IntersectionClass` and `OWLClass` connects the classes of an intersection. Associations `EquivalentClass` and `DisjointClass` represent the OWL class axioms, e.g. `EquivalentClass` connects a class to another class with which it is defined to be equivalent.

The class `OWLRestriction` is defined as a subclass of `OWLClass`. OWL distinguishes two kinds of property restrictions: value constraints and cardinality constraints. All OWL property restriction types are defined as subclasses of the class `OWLRestriction`. A restriction class should have exactly one property `OWLonProperty` linking the restriction to a particular property. The restriction class must also have a property that represents the value or cardinality constraint on the property under consideration.

OWLBase Package - Properties. As shown in Figure 3, the OWL metamodel refines the `RDFProperty` class to support specific OWL properties. Both object properties and datatype properties can be declared as "functional". For this purpose, we define the class `FunctionalProperty` as a special subclass of the class `Property`. `Property` is an abstract class that simplifies representation of property equivalence and deprecation, simplifies constraints for OWL DL and OWL Full, and facilitates mappings with other metamodels.

The class `InverseFunctionalProperty` is a subclass of `OWLObjectProperty`, since only object properties can be declared to be inverse functional. A property is defined as symmetric or transitive by making it an instance of the class `SymmetricProperty` or `TransitiveProperty` respectively, both defined as subclasses of `OWLObjectProperty`. Equivalent and inverse properties can be specified with the associations `EquivalentProperty` and `InverseProperty`.

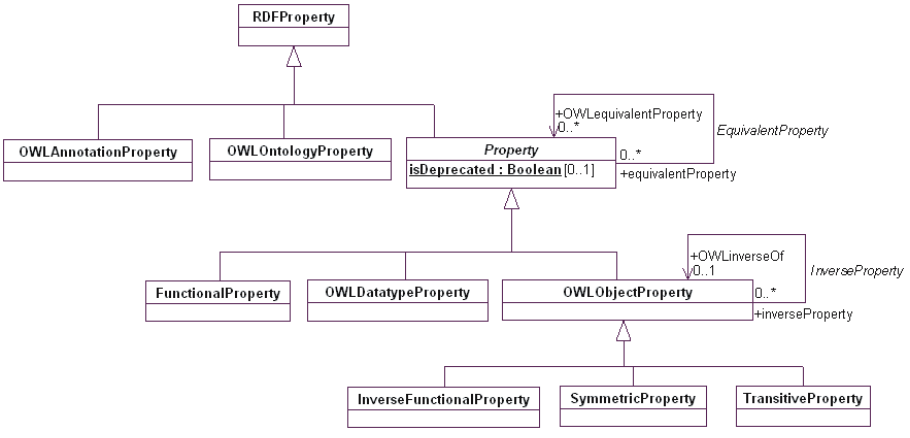

Fig. 3. The OWL Properties Diagram

OWLBase Package - Individuals. Individuals are represented in a subclass `Individual` of the class `RDFSResource`. OWL does not make the so-called unique name assumption. For the statements that two individuals are different or the same, the ODM has two associations `DifferentIndividual` and `SameIndividual` connected to the class `Individual`. The OWL construct owl: AllDifferent is represented by a subclass of `OWLClass`, the class `OWLAllDifferent`, for which the property `DistinctIndividuals` is defined to link an instance of OWLAllDifferent to a list of `Individuals`.

OWLBase Package - Datatypes. OWL makes use of the RDF datatyping scheme and provides an additional construct, `OWLDataRange`, for defining a range of data values, namely an enumerated datatype. It makes use of the owl:oneOf construct. The subject of `OWLoneOf` is an anonymous node of class `OWLDataRange` and the object is a list of `RDFSLiterals`.

OWLBase Package - OWL Universe. In Figure 4, we provide the part of the metamodel which facilitates ontology traversal for mapping purposes as well as utility in defining constraints for distinguishing OWL DL and OWL

Full. The class `OWLUniverse` specifies the set of ontology elements (i.e. classes, individuals, and properties) that together comprise a particular OWL ontology. It is intended to simplify packaging/mapping requirements for cases where the ability to determine the set of all elements is required.

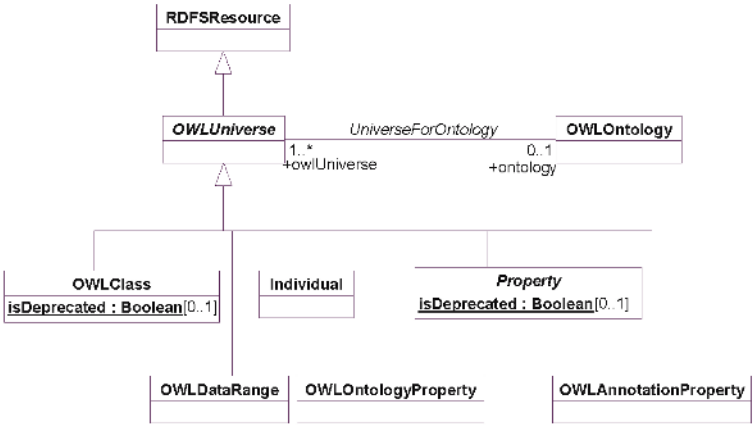

Fig. 4. The OWL Universe Diagram

OWLDL and OWLFull Package. The OWLBase package we just described supports the constructs common to both OWL DL and OWL Full. We provide two additional subpackages to distinguish between the two dialects. Both consist of either extensions or constraints on the OWLBase package. Users can use either or both of the subpackages together with the OWLBase package, depending on whether they want to work with OWL DL or OWL Full. For a complete listing of OWLDL and the OWLFull package, we refer the reader to Sections 11.8 and 11.9 of [13]. An extract of them is given here.

Some of the constraints in the OWLDL package are:

- The set of classes, datatypes, datatype properties, object properties, annotation properties, ontology properties, individuals, data values, and other built-in vocabulary are pairwise disjoint.
- All classes and properties must be explicitly typed as class respectively properties.
- Axioms about individual equality and difference must be about named individuals only (a consequence of category separation).

The OWLFull package contains additional extensions to support the lack of disjointness between classes, properties and individuals. In particular, these extensions provide additional attributes on the OWLBase metamodel classes as

well as definitions of new intersection classes required as a workaround to implement OWL Full. The need for this workaround results from a limitation in the MOF2 instances model, which requires that an InstanceSpecification be associated with exactly one classifier. This makes it impossible to have an object as an instance both of Individual and OWLClass, for example. When a future revision of MOF relaxes the instances model to permit multiple classifiers, the OWLFull Package will become superfluous.

3.2 A UML Profile for OWL Ontologies

Our UML profile is designed to support modelers developing ontologies in OWL through reuse of UML notation using tools that support UML2 extension mechanisms. The profile reflects the structure of the OWL metamodel (and the OWL language). We reuse the standard UML2 notation when the constructs have the same intuitive semantics as OWL, or, when this is not possible, stereotyped UML constructs that are consistent and as close as possible to OWL semantics. Stereotypes are leveraged extensively and are represented as the OWL metaclass names enclosed in `'<<...>>'`. In the following, we introduce our UML2 profile for OWL ontologies. We focus on property representation and refer the reader to Chapter 14 of [13] for a full account. First, we represent the constructs for RDF properties, since the OWL profile package imports the RDF profile package. Then, we show how we refine these RDF property constructs for OWL. We provide considerable flexibility so that property representation is truly intuitive for those familiar with UML.

In UML, a property can be defined as part of an association or on the class that defines the domain of the property. In this case the type of the property is the class that defines its range. When a property is part of an association, the association is binary with unidirectional navigation, from the class that defines the domain of the property to the class that defines its range. In RDF and OWL, properties are defined globally, that is, they are available to all classes in all ontologies. For RDF properties that are defined without specifying a domain or range, the profile uses a global `Thing` class (`Thing` for RDF/S, `owl:Thing` in OWL ontologies) as default for the Ş̌missingŤ end class. Properties that are defined with such a default domain or range may not have multiplicities (other than [0..*]) or other constraints that correspond to OWL restrictions. Figure 5 shows an example of a property without a specified domain. From a UML perspective, properties are semantically equivalent to binary associations with unidirectional navigation (Ş̌one-wayŤ associations). Figure 6 shows the alternate representation for properties. Just like a UML property, there is efficient navigation from an instance of `Thing` to an instance of `Color` through the `hasColor` end. Moreover, associations can be classes, as shown in Figure 7. An association class can have properties, associations, and participate in generalization as any other class. Notice that the association has a (slightly) different name than the property, by capitalizing the first letter, to distinguish the association class from the property itself. A stereotype `<<rdfProperty>>` is introduced to highlight such binary, unidirectional association classes, as shown in the Figure.

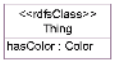

Fig. 5. Property `hasColor` without specified domain

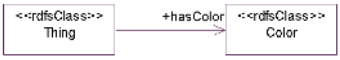

Fig. 6. Property `hasColor` without specified domain - alternate representation

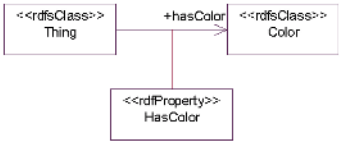

Fig. 7. Property `hasColor` - association class representation

The representation of RDF/S and OWL property subtyping (i.e., rdfs:subPropertyOf) is depending on which of the three notations above is used. In case of the UML property representation (Figure 5), we add a second property entry in the class, and use subsetting by adding `{subsets <super-property-name>}` at the end of that property entry. For the unidirectional association (Figure 6), we add another association for the subproperty, and add `{subsets <super-property-name>}` to the association. In case of the association classes (Figure 7), a UML generalization with the stereotype `<<rdfsSubPropertyOf>>` is preferred. For specific OWL properties, we use stereotypes like `<<objectProperty>>` instead of `<<rdfProperty>>`. In these properties, additional characteristics, e.g. a property being functional or a property being symmetric, are represented as UML properties.

If users want to specify a `owl:equivalentProperty` or `owl:inverseOf` relation between two properties, the notation is quite straightforward as well. For instance, Figure 8 shows an `owl:inverseOf` relation being modeled between two association classes using an `<<inverseOf>>` stereotype. An arrowhead is used opposite from the association class that will have owl:inverseOf in XML syntax.

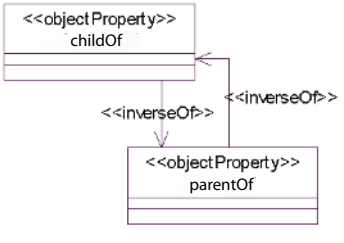

Fig. 8. Using owl:inverseOf Between Association Classes

3.3 Mappings Between UML and OWL

This Section introduces mappings to transform models between OWL and UML, based on the metamodel and the profile described in the previous sections. The ODM Request for Proposals (RFP [19]) called for a normative mapping between the single, unified Ontology Definition Metamodel originally envisioned and UML. If a such a single, normative mapping were provided, for a given implementation to be considered compliant, it would necessarily support that exact mapping. Over the course of development of the ODM, we determined that restricting our potential user community to any specific dialect of OWL (Lite, DL, or Full) would not support the long term vision we outlined in the usage scenarios given in Chapter 7 of the specification. Any single, normative mapping would necessarily force adherence to a specific dialect of OWL.

That said, we claim that the mappings given in the specification can be very informative, and are included in the specification for a number of reasons. First, they demonstrate feasibility of mapping in general and implement one set of design choices, providing a baseline from which a particular implementation can vary. Second, they bring clearly to the fore the detailed relationships among the metamodels. These relationships can help those who understand one of the target languages to come to an understanding of the others. Finally, for many applications, particularly lighter weight vocabularies and ontologies, the mapping provided is sufficient to support transformations between OWL and equivalent UML models, which remains a primary goal of the ODM.

Table 1. Feasible Mappings between UML and OWL

UML Feature	**OWL Feature**	**Comment**
class, type	class	
instance	individual	
ownedAttribute, binary association	property, inverseOf	
subclass, generalization	subclass, subproperty	
N-ary association, association class	class, property	Requires decomposition
enumeration	oneOf	
disjoint, cover	disjointWith, unionOf	
multiplicity	minCardinality, maxCardinality, FunctionalProperty, InverseFunctionalProperty	OWL cardinality restrictions declared only for range
package	ontology	

Table 1 provides a very high level summary comparison of some features of UML giving the equivalent OWL feature. UML features are grouped in clusters

that translate to a single OWL feature or a cluster of related OWL features. The mapping itself, as described in Chapter 16 of [13], reflects transformation of a model represented in the ODM metamodels for RDF and OWL to the corresponding UML metamodel element(s), and is informed by the profile(s) given in Chapter 14. The representation given in the specification includes both explanatory text and a formal mapping expressed in the recently adopted MOF Query/Views/Transformations (QVT) language [19], which provides a standardized MOF-based platform for mapping instances of MOF metamodels from one metamodel to another. The mapping provided is explicitly between UML 2 and the DL dialect of OWL. For a full account of the informative mappings and their formal expressions in QVT, we refer to [13].

4 Implementation and Examples

This section demonstrates two implementations which have been developed in the context of the ODM submission at OMG: the Visual Ontology Modeler and the Integrated Ontology Development Toolkit[1].

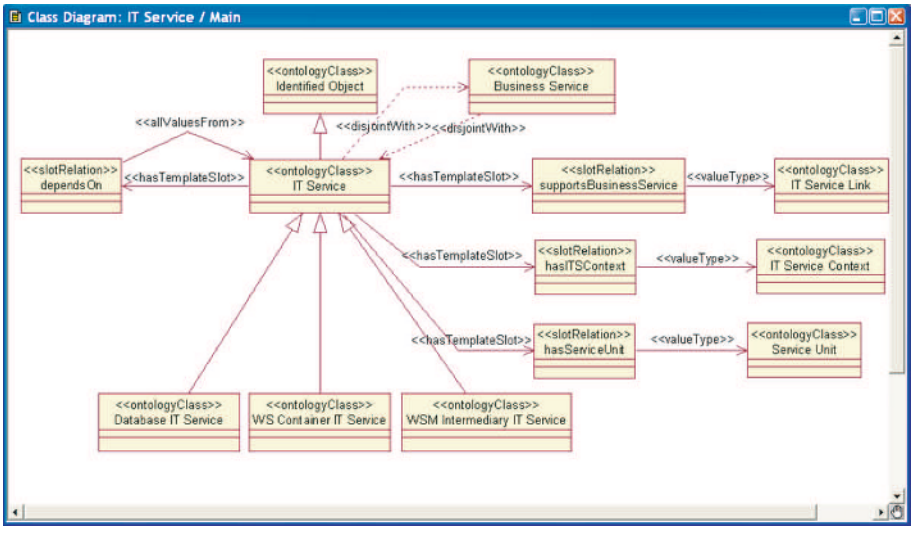

Fig. 9. A diagram modeled with the VOM tool

[1] Commercial equipment and materials might be identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the U.S. National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Visual Ontology Modeler. Visual Ontology Modeler (VOM), developed at the company Sandpiper, is currently implemented as an add-in to IBMŠs Rational Rose product. The current release is compatible with our ODM metamodels and profile for RDFS/OWL. A library of ontology components including ontologies representing several metadata and ISO standards are available for use with the tool. VOM supports forward and reverse engineering of RDFS/OWL ontologies and import/export of ODM/XMI ([22]) (and thus of any MOF metamodel or UML model that can be transformed to ODM/XMI). VOM users have demonstrated measurable productivity gains in ontology development and maintenance as well as increased consistency in RDFS/OWL generation for new and existing ontologies. Figure 9 shows a simple ontology fragment for management application integration ([17]) modeled using VOM (for lack of space we do not show a full screenshot). The second-generation VOM, which is currently in development, will support IBMŠs Eclipse ([9]) and Eclipse Modeling Framework (EMF, [6]) based modeling environment. An open-source version of the software that provides basic functionality will be available for EMF users.

Integrated Ontology Development Toolkit. The EMF-based IBM Integrated Ontology Development Toolkit (IODT) is a toolkit for ontology-driven development, including an EMF Ontology Definition Metamodel ([25]) (EODM[2], based on our ODM), an Eclipse-based ontology-engineering environment, and an OWL ontology repository, which has been evaluated to be highly scalable and perform better than several other well-known systems [16]. The toolkit supports RDFS/OWL parsing and serialization, TBox and ABox reasoning, transformation between RDFS/OWL and other data-modeling languages, and SPARQL[3] query. This toolkit has over 1,800 downloads in alphaWorks and Eclipse.

5 Related Work

In recent years, an increasing range of software systems engage in a variety of ontology management tasks, including the creation, storage, search, query, reuse, maintenance, and integration of ontologies. Recently, there have been efforts to externalize such ontology management burden from individual software systems and put them together in middleware known as an ontology management system. However, as far as we know, other proposals based on the visual UML and MOF ([2], [3], [4], [10]) provide an approach with some similarities and some different design considerations as well, but no full implementation. [2], [3] and [4] are currently being merged with our solution.

6 Conclusion and Future Investigations

We presented a MOF based metamodel and a respective UML profile for OWL DL and OWL Full. Furthermore, we provided feasible mappings which support

[2] http://www.eclipse.org/emft/projects/eodm/
[3] http://www.w3.org/TR/rdf-sparql-query/

the transformation between OWL ontologies and UML models and vice versa. This enables ontology engineers to build OWL ontologies based on UML using existing UML tools. Considering the amount of people familiar to UML, our solution will be an good approach to ontology modeling for ordinary developers. With the ODM defined in MOF, we can further utilize MDA's support in modeling tools, model management and interoperability with other MOF-defined metamodels. We expect that the interoperability with existing software tools and applications will ease ontology development and thus contribute to the adoption of semantic technologies and their success in real-life applications.

We have implemented our approach to validate our ideas in the Visual Ontology Modeler and the Integrated Ontology Development Toolkit.

Next to finishing and evaluating the ODM submission in the near future, we plan to extend the ODM to facilitate the development of rules as well. Which rule formalisms we will eventually support, is heavily depending on the outcome of the Rule Interchange Format working group at W3C ([26]). Some initial work on a metamodel and UML Profile for rules is presented in [2].

References

1. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, W3C, February 2004. W3C Recommendation.
2. S. Brockmans, P. Haase, P. Hitzler, and R. Studer. A Metamodel and UML Profile for Rule-extended OWL DL Ontologies. In *3rd Annual European Semantic Web Conference*, Budva, Montenegro, June 2006. Springer.
3. S. Brockmans, P. Haase, and H. Stuckenschmidt. Formalism-Independent Specification of Ontology Mappings - A Metamodeling Approach. In *5th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2006*, Montpellier, France, November 2006.
4. S. Brockmans, R. Volz, A. Eberhart, and P. 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.
5. A. Brown. An introduction to Model Driven Architecture - Part I: MDA and today's systems, February 2004. `http://www-106.ibm.com/developerworks/rational/library/3100.html`.
6. F. Budinsky, R. Ellersick, T. J. Grose, E. Merks, and D. Steinberg. *Eclipse Modeling Framework*. The Eclipse Series. Addison Wesley Professional, first edition, 2003.
7. R. Colomb, K. Raymond, L. Hart, P. Emery, C. Welty, G. T. Xie, and E. Kendall. The Object Management Group Ontology Definition Metamodel. In F. Ruiz, C. Calero, and M. Piattini, editors, *Ontologies for Software Engineering and Technology*. Springer, 2006. to appear.
8. M. Dean and G. Schreiber. OWL Web Ontology Language Reference. Technical report, World Wide Web Consortium (W3C), Feb 2004. W3C Recommendation.
9. J. des Rivieres and W. Beaton. Eclipse Platform Technical Overview. July 2001. Updated April 2006 for Eclipse 3.1.
10. D. Djuric, D. GaŽevic, V. Devedđic, and V. Damjanovic. MDA Development of Ontology Infrastructure. In *Proceedings of the IADIS International Conference Applied Computing*, pages II–23–II–26, Lisbon, Portugal, 2004.

11. D. Frankel, P. Hayes, E. Kendall, and D. McGuinness. The Model Driven Semantic Web. In *The 1st International Workshop on the Model-Driven Semantic Web (MSDW 2004)*, Monterey, California, USA, September 2004. http://www.sandsoft.com/edoc2004/FHKM-MDSWOverview.pdf.
12. L. Hart, P. Emery, R. Colomb, K. Raymond, S. Taraporewalla, D. Chang, Y. Ye, E. Kendall, and M. Dutra. OWL Full and UML 2.0 Compared, March 2004. http:// www.itee.uq.edu.au/∼colomb/Papers/UML-OWLont04.03.01.pdf.
13. IBM and Sandpiper Software. Ontology Definition Metamodel. Sixth Revised Submission, Object Management Group, June 2006. http://www.omg.org/cgi-bin/doc?ad/2006-05-01.
14. ISO/IEC. Topic Maps Ũ Data Model. Technical Report 13250-2, December 2005.
15. ISO/IEC. Information technology – Common Logic (CL) - A framework for a family of logic-based languages. Technical Report 24707, April 2006. Official ISO FCD Draft.
16. L. Ma, Y. Yang, Z. Qiu, G. Xie, and Y. Pan. Towards A Complete OWL Ontology Benchmark. In *3rd Annual European Semantic Web Conference*, Budva, Montenegro, June 2006. Springer.
17. T. Nitzsche, J. Mukerji, D. Reynolds, and E. Kendall. Using Semantic Web Technologies for Management Application Integration. In *proceedings of the workshop on Semantic Web Enabled Software Engineering (SWESE)*, Galway, Ireland, November 2005. http://www.mel.nist.gov/msid/conferences/SWESE/accepted_papers.html.
18. Object Management Group. Ontology Definition Metamodel – Request For Proposal, March 2003. http://www.omg.org/docs/ontology/03-03-01.rtf.
19. Object Management Group. Revised submission for MOF 2.0 Query/Views/Transformations RFP. http://www.qvtp.org/downloads/1.1/qvtpartners1.1.pdf, August 2003.
20. Object Management Group. OCL 2.0 Specification. Technical Report Version 2.0, June 2005.
21. Object Management Group. Unified Modeling Language: Superstructure. Technical Report Version 2.0, August 2005.
22. Object Management Group. XMI Mapping Specification. Technical Report Version 2.1, September 2005.
23. Object Management Group. Meta Object Facility (MOF) Core Specification. Technical Report Version 2.0, January 2006. OMG Available Specification.
24. Object Management Group. Unified Modeling Language: Infrastructure. Technical Report Version 2.0, March 2006.
25. Y. Pan, G. Xie, L. Ma, Y. Yang, Z. Qiu, and J. Lee. Model-Driven Ontology Engineering. *In Journal of Data Semantics VII*, 2006. Springer.
26. W3C. Rule interchange format working group charter. http://www.w3.org/2005/rules/wg/charter, 2005.