JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2005**

Vol. 4, No. 1, January-February 2005

Ontology Modeling and MDA

Dragan Djurić, Dragan Gašević, Vladan Devedžić, University of Belgrade, Serbia and Montenegro

Abstract

The paper presents Ontology Definition Metamodel (ODM) that enables using Model Driven Architecture (MDA) standards in ontological engineering. Other similar metamodels are based on ontology representation languages, such as RDF(S), DAML+OIL, etc. However, none of these other solutions uses the recent W3C effort – The Web Ontology Language (OWL). In our approach, we firstly define the ODM place in the context of the MDA four-layer architecture and identify the main OWL concepts. Then, we define ODM using Meta-Object Facility (MOF). The relations between similar MOF and OWL concepts are discussed in order to show their differences (e.g. MOF or UML Class and OWL Class). The proposed ODM is a good starting point for defining an OWL-based UML profile that will enable using the well-known UML notation in ontological engineering more extensively.

1 INTRODUCTION

The Semantic Web and its XML-based languages are main directions of the future Web development. Domain ontologies [Gruber93] are the most important part of the Semantic Web applications. They are formal organization of domain knowledge, and in that way enable knowledge sharing between different knowledge-base applications. Artificial intelligence (AI) techniques are used for ontology creation, but those techniques are more related to research laboratories, and they are unknown to wider software engineering population.

In order to overcome the gap between software engineering practitioners and AI techniques, there are a few proposals for UML use in ontology development [Cranefield01]. But, UML itself does not satisfy needs for representation of ontology concepts that are borrowed from the Descriptive Logic, and that are included in Semantic Web ontology languages (e.g. RDF, RDF Schema, OWL, etc.). The OMG's Model Driven Architecture (MDA) concept has ability to create (using metamodeling) a family of languages [Duddy02] that are defined in the similar way like the UML is. Accordingly, in this paper, the authors define metamodel for ontology modeling language. This metamodel is defined using Meta-Object Facility (MOF), and is based on the Web Ontology Language (OWL).

Cite this article as follows: Dragan Djuric, Dragan Gasevic, Vladan Devedzic: "Ontology Modeling and MDA", in *Journal of Object Technology*, vol. 4, no. 1, January-February 2005, pp. 109-128. http://www.jot.fm/issues/issue_2005_01/article3

The overview of the Semantic Web languages and OWL is given in the next section, while the description of the MDA and MOF is in the section three. In the section four we give a framework for our approach of the ontology language metamodel in the MDA context. The ontology metamodel definition in detail is shown in the section five. Basing on the paper appendix, we give summary of the relations between OWL (as well as RDF and RDF Schema), ODM concepts, and Ontology UML Profile. The last section contains final conclusions. This work is a part of Good-Old-AI (http://goodoldai.org.yu) effort for developing AIR - a platform for intelligent information systems.

2 AN OVERVIEW OF THE SEMANTIC WEB AND THE WEB ONTOLOGY LANGUAGE

Before the expansion of the World Wide Web, there had been taken many different approaches to represent knowledge or create taxonomies. No matter how these approaches were good or bad, there was a significant problem – their interoperability. All these solutions were isolated, limited to single applications or to a homogenous group of applications, and oriented to a small domain. With the introduction of World Wide Web and, later, XML technologies, the infrastructure for sharing data in a common way emerged. However, current Web is only syntactically interoperable; it is designed for direct human processing of the meaning of data.

The next step in Web evolution is the Semantic Web [Berners99], which will enable machine-understandable data to be shared across the Net. The Semantic Web will be powered by metadata, described by ontologies that will give machine-understandable meaning to its data. Ontology is one of the most important concepts in knowledge representation. It can be generally defined as shared formal conceptualization of particular domain [Gruber93]. Ontologies are essential in the knowledge management systems, agent systems, e-commerce… These ontologies must be interconnectable enough to enable each ontology to be merged with others, creating one big encyclopedia that will be understandable to software agents without direct human interference. Thus, The World Wide Web and XML will provide the ontologies with interoperability, and these interoperable ontologies will, in return, facilitate Web that can “know” something.

Naturally, achieving such goal is a challenging task and path to Semantic Web will be longer than World Wide Web progress path was. The technology that has to enable such Web must be powerful enough and intelligent enough to accomplish inference about enormous quantities of data and yet be affordable and easy enough to use. That means that both powerful infrastructure and easy-to-use tools must be provided to potential users.

Common data interoperability in present applications is best achieved by using XML. XML (*eXtensible Markup Language*) is a meta-language used to define other languages. It describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them [Brickley00]. XML defines neither the tags nor grammar, which makes it completely extensible. It only

requires that document must be well-formed in a tree structure, so it could be parsed by standard XML tools. In addition, the document can be structured to be valid. A valid document is one that conforms to its XML Schema, which defines grammar and tag set for specific XML formatting.

Semantic Web architecture is a functional, non-fixed architecture [Berners98]. Barnes-Lee defined three distinct levels that incrementally introduce expressive primitives: *metadata* layer, *schema* layer and *logical* layer [Harmelen03].

While XML with XML Schema enables common, well-defined and easy processable syntax, it tells nothing about semantics of data it describes. That means that some standard must be built on top of XML that will describe semantics of data. The first step in that direction is Resource Description Framework (RDF), a general model in metadata layer and Resource Description Framework Schema (RDFS), language at schema layer. The RDF data model defines a simple model for describing interrelationships among resources in terms of named properties and values. RDF properties may be thought of as attributes of resources and in this sense correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources. As such, the RDF data model can therefore resemble an entity-relationship diagram [Berners98]. The RDF Schema declares these properties, and provides mechanisms for defining the relationships between these properties and other resources.

To enable reasoning services for the Semantic Web, another layer is needed on top of RDF(S). That (logical) layer introduces ontology languages, that are based on meta-modeling architecture defined in lower layer. It introduces a richer set of modeling primitives which can be mapped to Descriptive Logic. This enables using of tools with generic reasoning support, independent of specific problem domain. Common examples of such languages are OIL and DAML. The newest emerging standard is W3C's OWL.

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the World Wide Web. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL Web ontology language [Miller03]. The place of OWL in described architecture is shown on Figure 1.

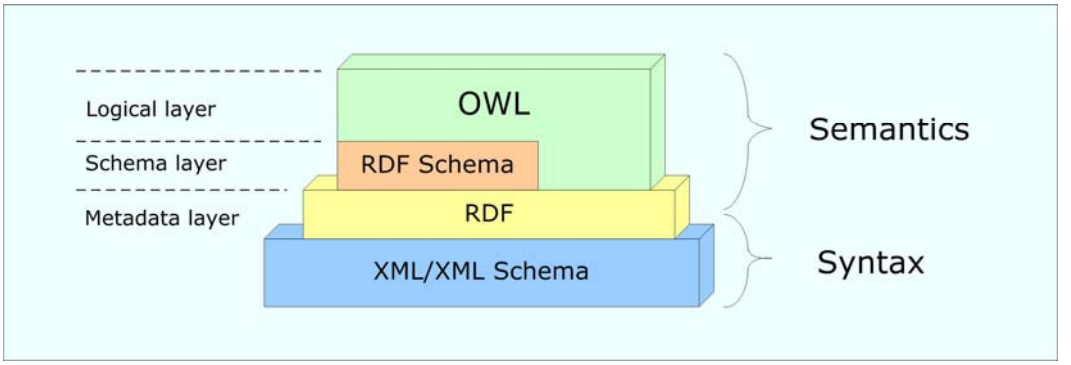


Figure 1. OWL in the Semantic Web architecture

Since World Wide Web is almost unconstrained, OWL must provide open world assumption and allow importing and mixing various ontologies. Some of them may be

even contradictory, but new information can never retract existing information, it can be only added to it. In order to provide such capabilities and, in the same time, to support calculations and reasoning in finite time with tools that can be built on existing or soon available technologies, OWL introduces three increasingly expressive sublanguages for various purposes: OWL Full, OWL DL and OWL Lite.

OWL Full provides maximal expressiveness and the syntactic freedom of RDF, but doesn't provide any computational guarantees. The main characteristic of OWL Full in comparison to OWL DL and OWL Lite is that one class, which is, by definition, a collection of individuals, can be the individual itself, like in RDF(S). It is obvious that this approach can lead to models that need infinite time to compute.

OWL DL (Descriptive Logic) enables maximal expressiveness and guarantees computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time). It includes all OWL Full constructs, and appends some constraints. The most significant constraints are that class cannot be an individual or property, or that property cannot be an individual or class. OWL DL has good formal background since it is contrived on descriptive logic.

OWL Lite is intended mostly to support classification hierarchy and simple constraint features. It is good starting point for tool builders. OWL Lite can be useful in migrations of existing taxonomies to OWL.

OWL Full is an extension of OWL DL, which is an extension of OWL Lite, thus every OWL Lite ontology is OWL DL and OWL Full ontology and every OWL DL ontology is OWL Full ontology.

3 AN OVERVIEW OF MODEL DRIVEN ARCHITECTURE AND META-OBJECT FACILITY

If we look back to the history of software development, we can see a notable increase of models abstraction. Modeling becomes more and more separated from underlying platform, making models of real world more reusable and easy to create by domain experts, requiring less knowledge of specific computer systems. This places software modeling closer to knowledge acquiring in knowledge engineering and vice versa. Current stage in that evolution is OMG's Model Driven Architecture [MOF02].

MDA defines three viewpoints (levels of abstraction) from which some system can be seen. From a chosen viewpoint, a representation of a given system (viewpoint model) can be defined. These models are (each corresponding to the viewpoint with the same name): *Computation Independent Model* (CIM), *Platform Independent Model* (PIM) and *Platform Specific Model* (PSM). CIM is a view of a system that does not show the details of a system structure. In software engineering it is also known as a domain model, which is concerned by domain experts. It is similar to a concept of ontology. PIM is model that is computation dependent, but it is not aware of specific computer platform details. In other words, it is targeted for technology-neutral virtual machine. Specification of

complete computer system is completed with PSM. The goal is to move human work from PSM towards CIM and PIM and let the specific platform detail implementations be generated as much as possible by automated tools which will do the transformation from PIM to PSM.

MDA is based on the four-layer metamodeling architecture, and several OMG's complementary standards; which is shown in figure 2. These standards are *Meta-Object Facility* (MOF) [Booch98], *Unified Modeling Language* (UML) [Booch98] and *XML Metadata Interchange* (XMI) [XMI02]. Layers are: meta-metamodel (M3) layer, metamodel (M2) layer, model (M1) layer and instance (M0) layer.

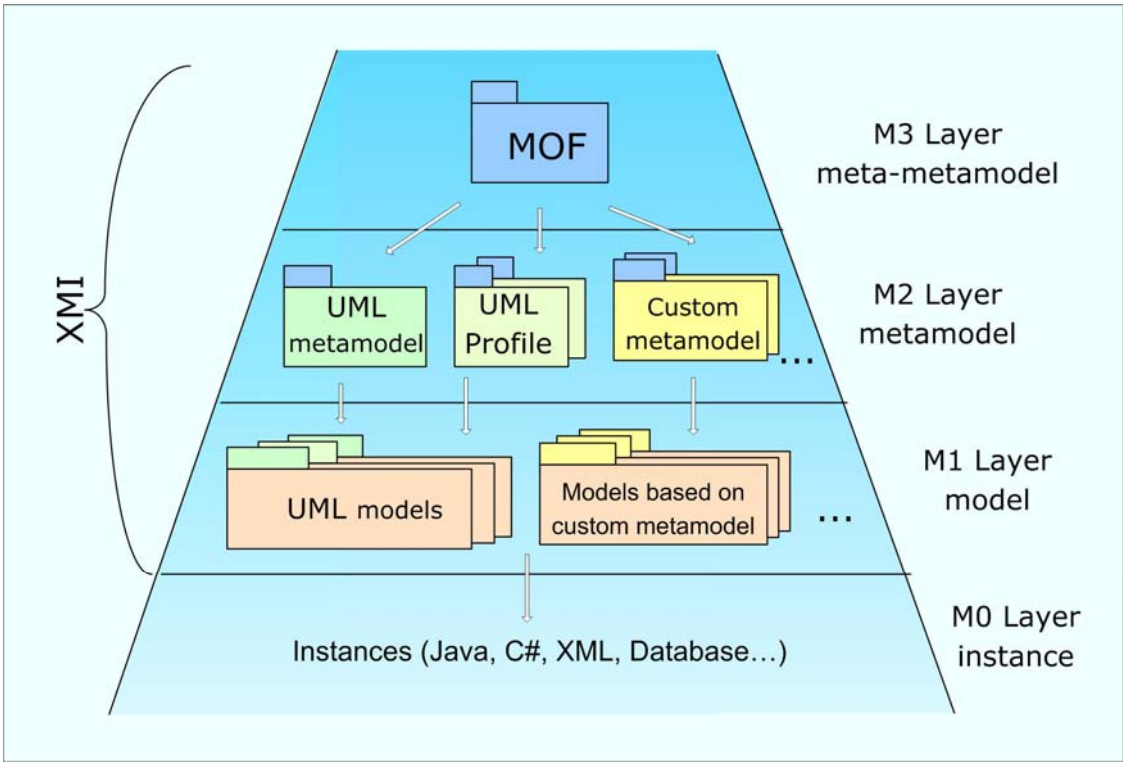


Figure 2. MDA four-layer MOF-based metadata architecture

On the top of this architecture is the meta-metamodel (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 or even MOF itself. MOF also defines a framework for implementing repositories that hold metadata (e.g. models) described by metamodels [Booch98]. The main aim of having four layers with common meta-metamodel is to support multiple metamodels and models; to enable their extensibility, integration and generic model and metamodel management.

All metamodels, standard or custom, defined by MOF are positioned on the M2 layer. One of these is UML, a graphical modeling language for specifying, visualizing and documenting software systems. With UML profiles, basic UML concepts (Class,

Association, etc.) can be extended with new concepts (stereotypes) and adapted to specific modeling needs. The models of the real world, represented by concepts defined in the corresponding metamodel at M2 layer (e.g. UML metamodel) are on M1 layer. Finally, at M0 layer, are things from the real world. An example would be: MOF Class (at M3) is used to define UML Class (M2), which is used to define model: Person (UML Class) and Tom, Dick and Harry (UML Objects) (M1), which represents reality (M0).

Another standard that is in base of this architecture is XMI, a standard that defines mapping from MOF-defined metamodels to XML documents and Schemas. XML, which has good software tools support, gives to XMI strength to enable solid shareability of meta-metamodel, metamodels and models.

Present software tools support for MDA is concentrated primarily on UML as a graphical notation, with no concern to metamodeling layers [Gasevic03]. UML CASE tools (e.g. Rational Rose, Borland Together, Magic Draw, Poseidon for UML, etc.) have good support for modeling at M1 layer and for programming languages code generation. Using appropriate UML profile they can generate databases, XML Schemas, EJBs etc. But, they lack support for M2 and M3 layers as well as unified serialization to XMI. It is expected from future tools to support UML 2, which will enable common XMI representation of UML models, and MOF-compliant model repositories at M2 and M3 layers, which will support metamodeling.

4 THE ONTOLOGY MODELING ARCHITECTURE

An overview

To be widely adopted by users and to succeed in real-world applications, knowledge engineering and ontology modeling must catch up with mainstream software trends. It will provide a good support in software tools and ease the integration with existing or upcoming software tools and applications, which will add values to both sides. To be employed in common applications, software knowledge management must be taken out of laboratories and isolated high-tech applications and put closer to ordinary developers. This issue has been addressed in more details in Cranefield's papers [Cranefield01].

MDA and its four-layer architecture provides a solid basis for defining metamodels of any modeling language, so it is the straight choice to define an ontology-modeling language in MOF. Such language can utilize MDA's support in modeling tools, model management and interoperability with other MOF-defined metamodels. Present software tools do not implement many of the concepts that are the basis of MDA. However, most of these applications, which are mostly oriented to the UML and M1 layer, are expected to be enhanced in the next few years to support MDA.

Currently, there is a RFP (Request for Proposal) within OMG that tries to define a suitable language for modeling Semantic Web ontology languages in the context of MDA [ODMRFP03]. According to this RFP the authors give their proposal of such

architecture. In our approach of ontology modeling in the scope of MDA, which is shown in Figure 3, several specifications should be defined:

- Ontology Definition Metamodel (ODM)
- Ontology UML Profile – a UML Profile that supports UML notation for ontology definition
- Two-way mappings between OWL and ODM, ODM and Ontology UML Profile and from Ontology UML Profile to other UML profiles.

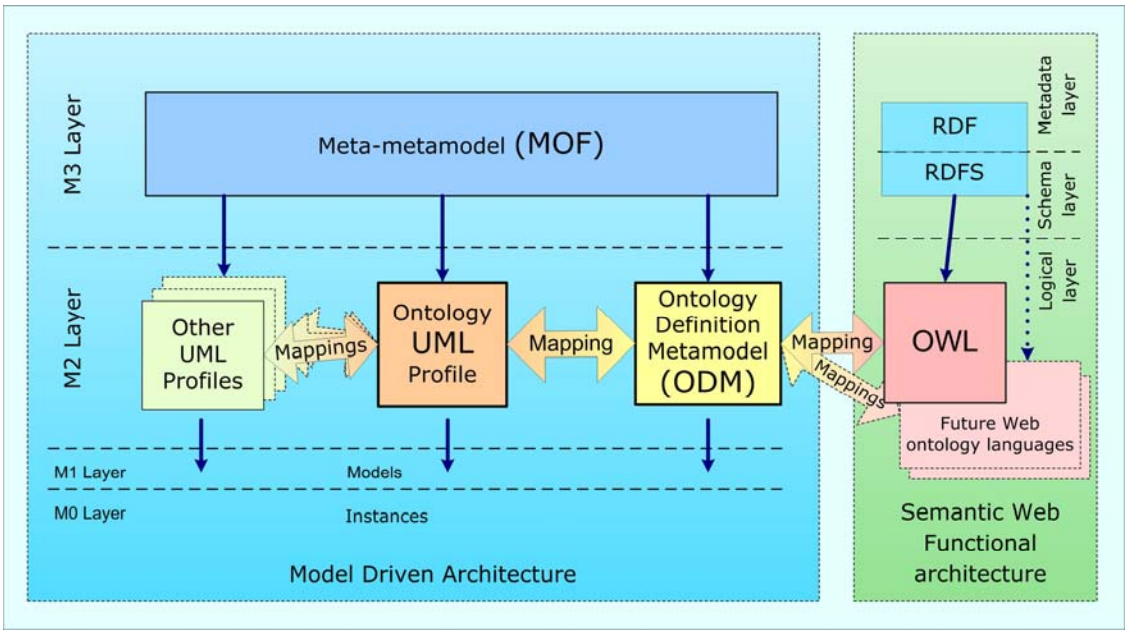


Figure 3. – Ontology modeling in the context of MDA and Semantic Web

Ontology Definition Metamodel (ODM) should be designed to comprehend common ontology concepts. A good starting point for ODM construction is OWL since it is the result of the evolution of existing ontology representation languages, and is going to be a W3C recommendation. It is at the Logical layer of the Semantic Web [Harmelen03], on top of RDF Schema (Schema layer). In order to make use of graphical modeling capabilities of UML, an ODM should have a corresponding UML Profile [Sigel01]. This profile enables graphical editing of ontologies using UML diagrams as well as other benefits of using mature UML CASE tools. Both UML models and ODM models are serialized in XMI format so the two-way transformation between them can be done using XSL Transformation. OWL also has representation in the XML format, so another pair of XSL Transformations should be provided for two-way mapping between ODM and OWL. For mapping from the Ontology UML Profile into another, technology-specific UML Profiles, additional transformations can be added to support usage of ontologies in design of other domains and vice versa.

Metamodeling: MDA vs. Functional Architecture

Before we start with more detailed description of ODM, we must clarify differences between metamodeling based on MDA, and functional architecture which is used for Web ontology languages definition. RDFS, as a schema layer language, has a non-standard and non-fixed-layer metamodeling architecture, which makes some elements in model have dual roles in the RDFS specification [Pan01]. Therefore, it is difficult to understand by modelers, lacks clear semantics (by assigning dual roles to some elements) and propagates "layer mistake" problem to languages it defines, in our case to OWL. MDA, on the other side, has fixed and well-defined four-layer architecture. It has separate metamodeling primitives on meta-metamodel and metamodel layer that are separated from ontology language (or some other MOF-defined language) primitives, which can have infinite layers, as in the case of OWL Full.

In OWL DL, functional architecture's problems are partially solved by introducing new modeling elements (`owl:Class` for example) that are used for defining ontologies. In this case, `rdfs:Class` is used only for defining `owl:Class`, `owl:ObjectProperty` and other ontology-modeling primitives. It is not used for modeling ontologies, which is done using ontology-modeling primitives. On the other hand, OWL Full allows unconstrained use of RDFS constructs, which means that it completely inherits RDFS' problems. ODM that supports OWL Full cannot be modeled directly using MOF if we want to preserve fixed-layer architecture.

Accordingly, ODM will be designed primarily to support OWL DL. Support for OWL Full will be included partially, for concepts that don't introduce significant problems or break fixed-layer architecture.

A brief comparative description of the most important metamodeling constructs in MOF and RDF(S), which will make reading the next sections easier, is shown in Table 1. Detailed description of MOF can be found in OMG's MOF specification document [Booch98]. RDF, RDFS and their concepts are described in detail in W3C documents [Berners98].

Table 1. A brief description of basic MOF and RDF(S) metamodeling constructs

MOF element	Short description	RDF(S) element	Short description
`ModelElement`	`ModelElement` classifies the elementary, atomic constructs of models. It is the root element within the MOF Model.	`rdfs:Resource`	Represents all things described by RDF. Root construct of majority of RDF constructs.
`DataType`	Models primitive data, external types, etc.	`rdfs:Datatype`	Mechanism for grouping primitive data.
`Class`	Defines a classification over a set of object instances by defining the state and behavior they exhibit.	`rdfs:Class`	Provides an abstraction mechanism for grouping similar resources. In RDF(S),

`Classifier`	Abstract concept that defines classification. It is specialized by `Class`, `DataType`, etc.		`rdfs:Class` also have function that is similar to a MOF concept of `Classifier`.
`Association`	Expresses relationships in the metamodel between pairs of instances of `Class`es	`rdf:Property`	Defines relation between subject resources and object resources.
`Attribute`	Defines a notional slot or value holder, typically in each instance of its `Class`.		
`TypedElement`	The `TypedElement` is an element that requires a type as part of its definition. A `TypedElement` does not itself define a type, but is associated with a `Classifier`. Examples are object instances, data values etc.		In RDF(S), any `rdfs:Resource` can be typed (via the `rdf:type` property) by some `rdfs:Class`

5 ESSENTIAL ODM CONCEPTS

Resource

OWL is built on top of RDF; thus it inherits its concepts, such as Resource, Property, metamodeling capabilities etc. Resource is one of the basic RDF concepts; it represents all things described by RDFS and OWL. It may represent anything on the Web: a Web site, a Web page, a part of a Web page, or some other object named by URI. Compared to ontology concepts, it can be viewed as a root concept, the Thing. In RDFS, Resource is defined as an instance of `rdfs:Class`; since we use MOF as a meta-metamodeling language, Resource will be defined as an instance of MOF `Class`. It is the root class of most other basic ODM concepts that will be described: Ontology, Classifier, Property, Instance etc. The root of this hierarchy is shown on Class Diagram in Figure 4. Other class diagrams (shown in figures 5, 6 and 7) will depict these concepts in more detail.

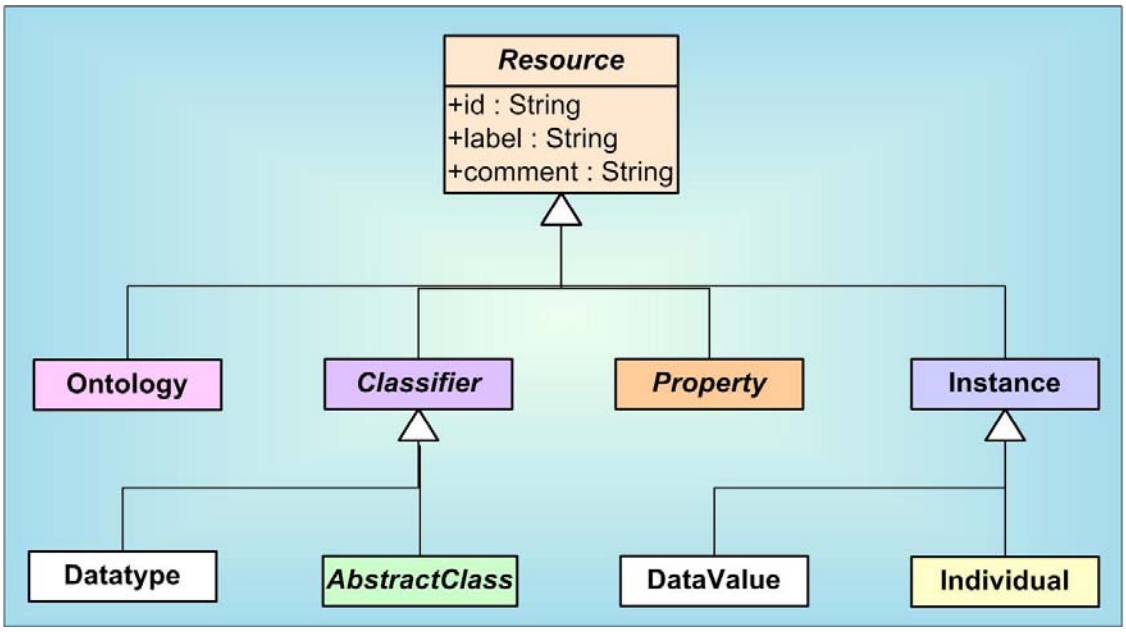


Figure 4. The hierarchy of basic ontology concepts

`Ontology` is a concept that aggregates other concepts (`Classes`, `Properties`, etc.). It groups instances of other concepts that represent similar or related knowledge. `Classifier` is the base class of concepts that are used for classification – `AbstractClass` and `DataType`. `Instance` is the base class of concepts that are classified by `Classifiers` – concrete `Individuals` and concrete `DataValues`. `Property` is used to represent relationships between other concepts.

For example, `Person` is an `AbstractClass` (more precise - a `Class`) that classifies many `Individuals: Tom, Dick, Harry` etc. All `Persons` have some characteristics – name and occupation, which are represented by `Properties – name` and `occupation.` These `Properties` can have values that are of certain type; `name` can be a `String` (an example of `DataType`), `occupation` can be `Profession` (another example of `AbstractClass`). Then, `Profession` classifies concrete professions (its `instances`): `Musician, Writer, Mechanic, Astronaut`…

Classifier

In RDFS and OWL, Class (`rdfs:Class` and `owl:Class`) represents a concept for grouping resources with similar characteristics. This concept of Class (we can also call it Ontology Class) is not completely identical as a concept of Class that is defined in UML and object oriented programming languages. Every `owl:Class` is a set of individuals, called class extension. These individuals are instances of that class. Two classes can have the same class extension but still be different classes. Ontology classes are set-theoretic, while traditional classes are more behavioral. Unlike a traditional class, an OWL class

does not directly define any attributes or relations with other resources, and there is no any concept similar to methods. Attributes and relations are defined as Properties. In ODM, a Class concept corresponding to `rdfs:Class` is defined as Classifier - an instance of `MOF Class` that inherits `Resource`. A concept that complies with `owl:Class` is ODM's `AbstractClass`.

OWL further introduces six ways of defining a `Class` – class descriptions:

- A class can be defined by a class identifier (an URI reference) – For example, a `Class Person`.
- As an exhaustive enumeration of individuals that form the instances of a `Class`. For example, individuals `Mick, Keith, Ron, Bill` and `Charlie` form an `Enumeration – TheRollingStones.` Note that they are also members of a `Class Person`.
- As a property restriction – `Class` of all individuals that have the same restriction on some of their characteristics.
- As an intersection – A `Class` of all individuals that are members of all `Classes` that form an intersection. An intersection of `Classes TheWailers` and `TheRollingStones` is a `Class` that does not have any member, since no musician has played in both bands.
- As a union – A `Class` of all individuals that are members of any Class that forms a union. A union of `TheWailers` and `TheRollingStones`, has twelve individuals, all musicians from both bands.
- As a complement – A `Class` of all individuals that are not members of other, complement class. A complement of `TheRollingStones` is a `Class` that has about six billion members – all Persons that are not members of `TheRollingStones.`
- `AllDifferent` is a helper class, which states that all of its instances are have different identity.

The first concept, named class is modeled as ODM `Class`. Other five species are defined in OWL as subclasses of `owl:Class`, and are shown in Figure 5.

If we define class descriptions as simple subclasses of `Class`, like it is defined in OWL, we will have some problems related to the differences between RDFS and MOF concept of a class and the open-world assumption of the Semantic Web. While in RDFS some class instance can be easily defined to be a member of many class extensions in the same time, in MOF it can be instance of exactly one class. The open-world assumption might demand some flexibility, i.e. that class which was a `Union` becomes an `Intersection`, which is not possible to model in MOF, since each instance can be the instance of only one `Class`, i.e. dynamic classifiers are not allowed.

To solve this problem, we used the idea captured in the *Decorator* design pattern [Gamma95]. In Figure 5, we define `ClassDescription` as a subclass of `Class` which can encapsulate a `Class`. In that way, we can have a chain of additions to the

starting definition of `Class` (i.e., speaking in software engineering terms, we can add further responsibilities to the original concept of `Class`). For example, if we have some simple `Class`, we can define union by decorating that class with `Union`, and change it later to intersection, by removing the union decorator and decorating the class with `Intersection`.

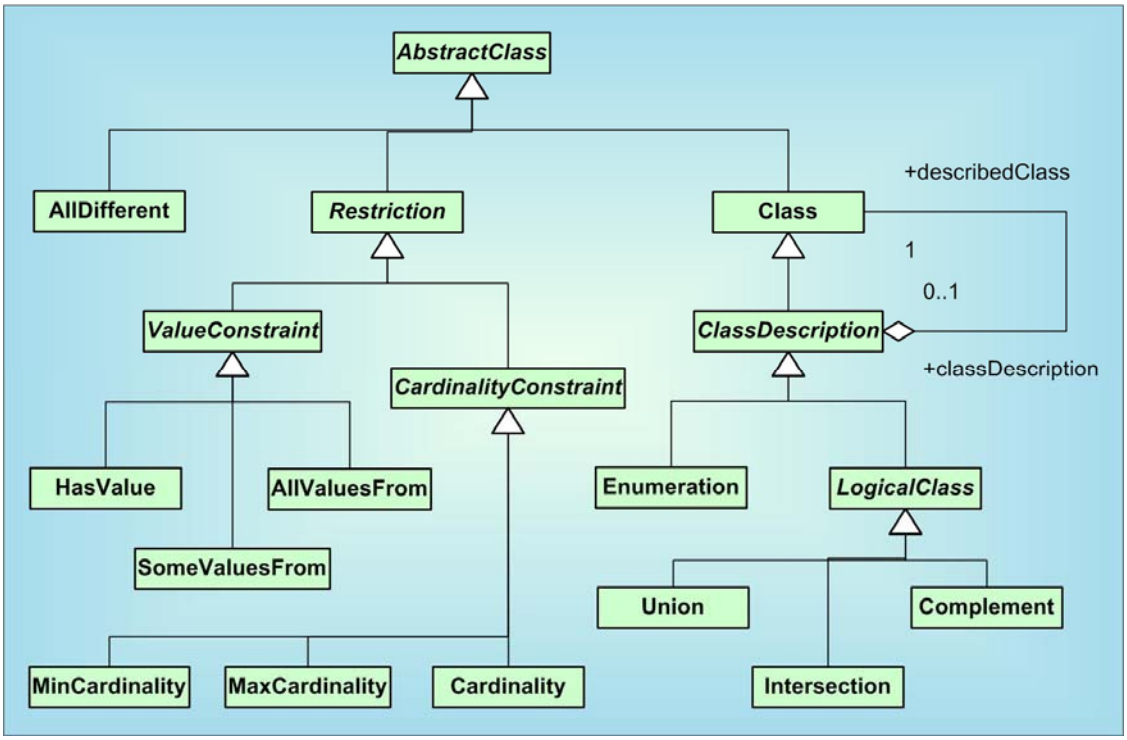


Figure 5. The hierarchy of Ontology Classes in ODM

Property

Ontology Class attributes or associations are represented through properties. A property is a relation between a subject resource and an object resource. Therefore, it might look similar to a concept of attribute and association in traditional, object oriented sense. However, the important difference is that Property is stand-alone; it does not depend of any Class (or resource) as associations or attributes are in UML. In ontology languages, a property can be defined even with no classes associated to it. In ODM, `Property` is an instance of MOF `Class` that inherits `Resource`.

In addition to the concept of `rdf:Property`, which is defined in RDF, OWL distinguishes two types of properties: `owl:ObjectProperty`, whose range can be only an `Individual`, and `owl:DatatypeProperty`, whose range can be only `DataValue`. In ODM, these concepts are instances of MOF `Class` that inherit `Property`. OWL also defines additional concepts, global cardinality constraints on a `Property` that can further refine the `Property`. These concepts are also represented as instances of MOF `Class`.

In OWL, various types of global property constraints are defined as subclasses of `Property`. Here we have the same problem we had with OWL classes, since some property might have multiple global constraints, for example symmetric and transitive. In this case we also apply the Decorator design pattern, just like we did with Class Descriptions. The resulting class diagram is shown in Figure 6. If we want to define, for example, symmetric property, we will decorate `ObjectProperty` with `SymmetricProperty`, and if we later decide that this property also should be transitive, we can simply decorate it again with `TransitiveProperty.`

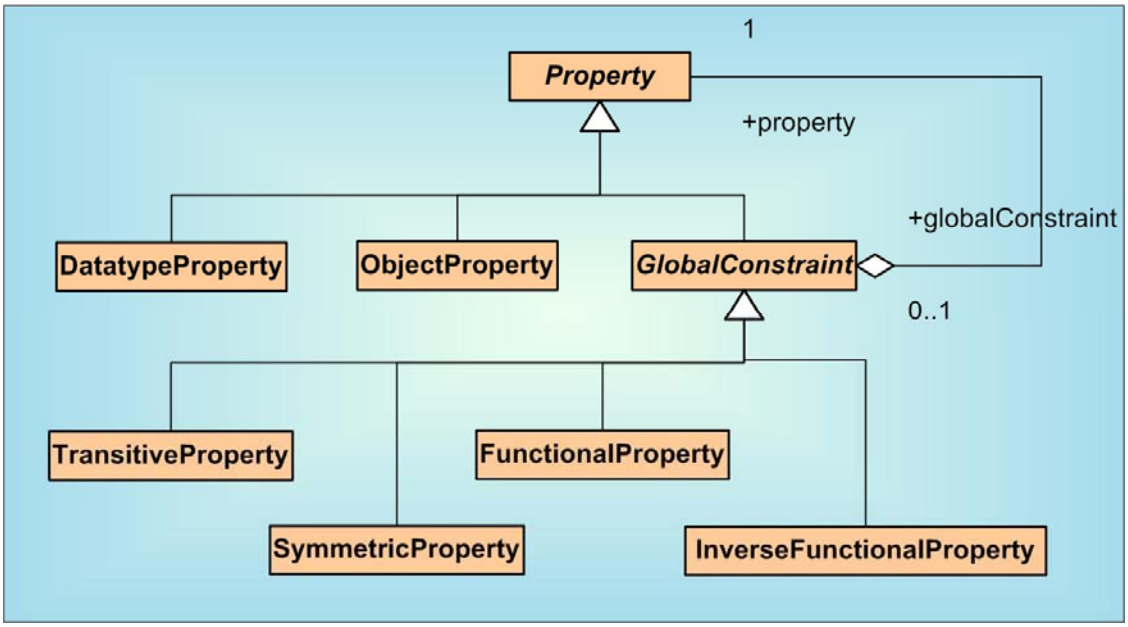


Figure 6. The hierarchy of Ontology Properties in ODM

Properties predefined in RDFS and OWL

We have seen how predefined concepts, which are defined in OWL as instances of `rdf:Class`, are defined in ODM as instances of MOF `Class` with some changes in the hierarchy. RDF(S) and OWL have some predefined concepts that are instances of `rdf:Property`. These predefined properties are used to make relationships between concepts in OWL metamodel. In ODM, they are modeled as MOF `Associations` or as MOF `Attributes`.

Predefined properties of RDF(S) and OWL and their ODM counterparts are not completely identical. For example, the predefined property `rdf:type` states that a `rdfs:Resource` is an instance of a `rdfs:Class`. In ODM, it is represented as an `Association` between `Classifier` and `Instance`, as shown in Figure 7, which is obviously a narrower usage than is defined in RDF. Recall that `Classifier` is further specialized in `AbstractClass` and `DataType`, and that `Instance` is specialized in `Individual` and `DataValue`. Such differences are caused by differences between MDA and Functional architecture. In RDF, `rdf:type` property is used as both

metamodeling and modeling concept while in MDA, MOF is used for metamodeling, and ODM for modeling. Since ODM type association is not used for metamodeling, it is a narrower concept than `rdf:type`, thereby they are not equal.

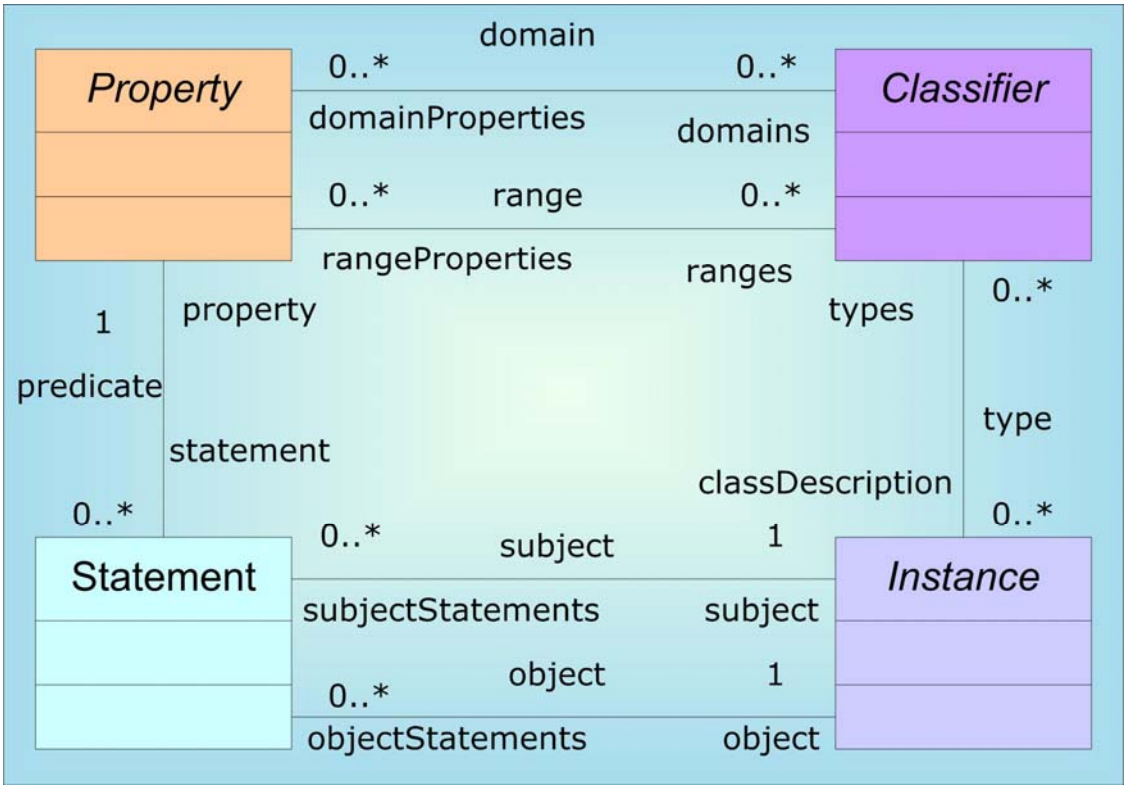


Figure 7. Key relationships among Ontology concepts

Example of predefined property that is modeled as a MOF Attribute is shown in Figure 4, as each of `Resource`'s attributes `ID`, `comment` and `label`.

A `Classifier` describes some general concept that has its `Instances` (`Individuals` and `DataValues`). On the other hand, a `Property` describes some generic characteristic that can describe that `Classifier` and possibly other `Classifiers`. Through `domain` we state that a Property can be used to describe a `Classifier`, and through `range` a characteristic's type. For example, a `Property nationality` can be assigned to a `Class Person` (through `domain`) with possible values which type is a `Class Country` (through `range`). In ODM, these relations are modeled as associations, as shown in Figure 7.

It is obvious that an `Individual` cannot have a `DataType` as its type, or that a `DataValue` cannot have an `AbstractClass` as its type. Looking at this class diagram, we can not see this constraint. Such constraints are described in the Object Constraint Language (OCL) [Booch98], a standard way of defining constraints in MOF

and UML. For example, to state that type of an `Individual` must be an `AbstractClass`, we add the following OCL constraint:

```
context: Individual
inv: self.type.oclIsTypeOf(AbstractClass)
```

Statement

A Statement is a Subject-Predicate-Object triple that expresses some fact in a way similar to the way facts are expressed in English. A fact that some `Individual`, `Bob` for example, has some nationality, `Jamaican`, is expressed through a Statement, which links the `Instance Bob` as the *subject*, *the* `Property nationality` as the *predicate*, and the `Instance Jamaica` as the *object*. Thus, Statement can be viewed as some kind of `Property`'s instance. In ODM, `Statement` is an instance of MOF `Class` that is linked with `Instance` by *subject and object* associations and with `Property` by *predicate* association (Figure 7). ODM `Statement` slightly differs from the Statement defined in RDF (`rdf:subject` and `rdf:object` link `rdf:Statement` with `rdfs:Resource`). The difference arises from the fact that ODM is not intended for metamodeling as RDF is, similarly to the case with rdf:type

Summary of Ontology Definition Metamodel

The summary of ODM concepts is given in Table 1 in the Appendix. The first column represents original RDF, RDF(S) and OWL concepts, which are used as the starting point for defining the ODM. The corresponding ODM concepts are listed in second column. The third and fourth columns summarize the Ontology UML Profile, which will be described in our future work, and is given here for a brief overview.

6 CONCLUSIONS

The metamodel defined in this paper is in accordance with the OMG's RFP initiative for ontology modeling. Accordingly, we borrowed the name ODM for our metamodel from the OMG's RFP. The proposed solution enables using ontologies in the way that is closer to software engineering practitioners. Also, since the ODM is defined as a MOF-compliant language it is possible to store ontologies in MOF-based repositories, as well as to share and interchange ontologies using XMI.

The proposed ODM can be considered as a part of the effort to specify standard ontology metamodel. Its important feature is that it is based on OWL.

Future developments based on the proposed ODM include defining the ontology UML Profile. It should enable a wide use of UML notation in ontology modeling. This way, the ODM concepts can be used as stereotypes in the UML models (similar to UML CORBA Profile or other OMG's UML Profiles). Further plans include using Java

Metadata Interface (JMI) [Dirckze02] to enable creation, storage, access, discovery, and exchange of ODM-defined ontologies using standard Java interfaces.

REFERENCES

[Gruber93] Gruber, T. R., "A translation approach to portable ontology specifications", *Knowledge Acquisition*, Vol. 5, No. 2, 1993.

[Cranefield01] Cranefield, S., "UML and the Semantic Web", *In Proceedings of the International Semantic Web Working Symposium*, Palo Alto, 2001, www.semanticweb.org/SWWS/program/full/paper1.pdf.

[Duddy02] Duddy, K., "UML2 Must Enable A Family of Languages", *Communications of the ACM*, Vol. 45, No. 11, November 2002, pp 73-75.

[Berners99] Tim Berners-Lee, *Weaving the Web*, Orion Business Books, London, 1999.

[Bray00] Bray, T., et al (eds.), "Extensible Markup Language (XML) 1.0 (Second Edition)", *W3C Recommendation*, http://www.w3.org/TR/2000/REC-xml-20001006, 2000.

[Brickley00] Brickley, D., Guha, R.V. (eds.), "Resource Description Framework (RDF) Schema Specification 1.0", *W3C Candidate Recommendation*, http://www.w3.org/TR/2000/CR-rdf-schema-20000327, 2000.

[Berners98] Tim Berners-Lee, "Semantic Web Road Map", *W3C Design Issues*, http://www.w3.org/DesignIssues/Semantic.html, 1998.

[Harmelen03] van Harmelen, F., et al, "OWL Web Ontology Language Reference", *W3C Working Draft*, http://www.w3.org/TR/2003/WD-owl-ref-20030331/, 2003.

[Miller03] Miller, J., Mukerji, J. (eds.), "MDA Guide Version 1.0", *OMG Document: omg/2003-05-01*, http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf, May 2003.

[MOF02] "Meta Object Facility (MOF) Specification v1.4", *OMG Document formal/02-04-03*, http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf, April 2002.

[Booch98] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley, Massachusetts, 1998.

[XMI02] "OMG XMI Specification, v1.2", *OMGDocument formal/02-01-01*, http://www.omg.org/cgi-bin/doc?formal/2002-01-01, 2002.

[Gasevic03] Gasevic, D., Damjanovic, V., Devedzic, V., "Analysis of the MDA Standards in Ontological Engineering", *submitted for publication to the*

Sixth International Conference of Information Technology, Bhubaneswar, India, December 22-25, 2003.

[ODMRFP03] "Ontology Definition Metamodel Request for Proposal", *OMG Document: ad/2003-03-40*, http://www.omg.org/cgi-bin/doc?ad/2003-03-40, 2003.

[Sigel01] Sigel, J., "Developing in OMG's Model-Driven Architecture", Revision 2.6, *Object Management Group White Paper*, ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf, 2001.

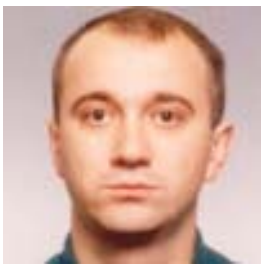
[Pan01] Pan, J., Horrocks, I., "Metamodeling Architecture of Web Ontology Languages", *In Proceedings of the First Semantic Web Working Symposium (SWWS'01)*, Stanford, July 2001, pp 131-149 http://img.cs.man.ac.uk/jpan/Zhilin/download/Paper/Pan-Horrocks-rdfsfa-2001.pdf.

[Gamma95] Gamma, E., et al, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[Dirckze02] Dirckze, R. (spec. leader), "Java Metadata Interface (JMI) Specification Version 1.0", http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html, 07 June 2002.

About the authors

Dragan Djurić is a PhD candidate at FON – School of Business Administration, University of Belgrade, and also a member of Good-Old-AI research group. His interests mostly include Enterprise software architecture, Object-Oriented development, Java platform and Intelligent Information Systems. He can be reached at dragandj@gmail.com.

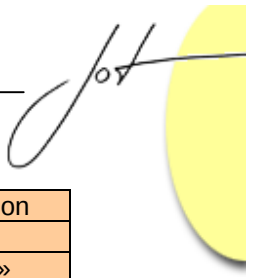


Dragan Gašević is a lecturer of Computer Science with the Military academy, Belgrade Serbia and Montenegro, as well as a researcher with the GOOD OLD AI research group, University of Belgrade. He has received his BS, MS, and PhD degrees in computer science from the University of Belgrade in 2000, 2002, and 2004, respectively. His research interests mostly include Semantic Web, ontologies, MDA, and applications of artificial intelligence techniques to education. Email: gasevic@yahoo.com.

Vladan Devedžić is an associate professor of computer science at the Department of Information Systems, FON - School of Business Administration, University of Belgrade, Serbia and Montenegro. His main research interests include software engineering, intelligent systems, knowledge representation, ontologies, Semantic Web, intelligent reasoning, and applications of artificial intelligence techniques to education and medicine. He can be reached at devedzic@fon.bg.ac.yu.

APPENDIX

RDFS Concept	Ontology Definition Metamodel Concept	Base UML Class	UML Stereotype (inside « ») or Tag
rdfs:Resource	abstract class Resource		
rdfs:Datatype	class Datatype	DataType	
rdfs:range	association range	Association or Attribute	«range»
rdfs:domain	association domain	Association or Attribute	«domain»
rdfs:type	association type	Dependency	«instanceOf»
rdfs:subClassOf	association subclassOf	Generalization	«subClassOf»
rdfs:subPropertyOf	association subPropertyOf	Generalization	«subPropertyOf»
rdfs:label	attribute label		
rdfs:seeAlso	association seeAlso	Association	«seeAlso»
RDF Concept	Ontology Definition Metamodel Concept	Base UML Class	UML Stereotype (inside «») or Tag
rdf:Property	abstract class Property		
rdf:Statement	class Statement	Object	«ObjectProperty» or «DatatypeProperty»
rdf:subject	association subject	Link or AttributeLink	«subject»
rdf:object	association object	Link or AttributeLink	«object»
rdf:predicate	association predicate	Dependency	«instanceOf»
rdf:ID	attribute ID	Element Name	
OWL Ontology Concept	Ontology Definition Metamodel Concept	Base UML Class	UML Stereotype (inside «») or Tag
owl:Ontology	class Ontology	Package	«ontology»
owl:Class	class Class	Class	«OntClass»
Enumeration	class Enumeration	Class	«Enumeration»



			or enumeration
owl:Restriction	abstract class Restriction		
owl:onProperty	association onProperty	Association	«onProperty»
ValueConstraint	abstract class ValueConstraint		
owl:allValuesFrom	association allValuesFrom and class AllValuesFrom	Association and Class	«allValuesFrom» (Assoc.) and «AllValuesFrom»
owl:someValuesFrom	association someValuesFrom and class SomeValuesFrom	Association and Class	«someValuesFrom» (Assoc.) and «SomeValuesFrom»
owl:hasValue	association hasValue and class HasValue	Dependency and Class	«hasValue» (Assoc.) and «HasValue»
CardinalityConstraint	abstract class CardinalityConstraint		
owl:minCardinality	class MinCardinality	AssociationEnd multiplicity	
owl:maxCardinality	class MaxCardinality	AssociationEnd multiplicity	
owl:cardinality	class Cardinality	AssociationEnd multiplicity	
owl:intersectionOf	association intersectionOf and class Intersection	Dependency and TaggedValue	«intersectionOf» (Dep.), intersection tag or «Intersection» for Class
owl:unionOf	association unionOf and class Union	Dependency and TaggedValue	«unionOf» (Dep.), union tag or «Union» for Class
owl:complementOf	association complementOf andClass ComplementOf	Dependency and TaggedValue	«complementOf» (for Dependency), complement tag or «Complement» for Class
owl:equivalentClass	association equivalentClass	Dependency	«equivalentClass»
owl:disjointWith	association disjointWith	Dependency	«disjointWith»
owl:ObjectProperty	class Objectproperty	Class	«ObjectProperty»
owl:DatatypeProperty	class DatatypeProperty	Class	«DatatypeProperty»
owl:equivalentProperty	association equivalentProperty	Dependency	«equivalentProperty»
owl:inverseOf	association inverseOf	Dependency	«inverseOf»
owl:FunctionalProperty	class FunctionalProperty	TaggedValue	functional
owl:InverseFunctionalProperty	class InverseFunctionalProperty	TaggedValue	inverseFunctional

owl:TransitiveProperty	class TransitiveProperty	TaggedValue	transitive
owl:SymmetricProperty	class SymmetricProperty	TaggedValue	symmetric
Individual	class Individual	Object	«ontClass»
owl:Thing	instance of class Individual		
owl:sameAs and owl:sameIndividualAs	association sameAs	Dependency	«sameAs»
owl:differentFrom	association differentFrom	Dependency	«differentFrom»
owl:allDifferent	association allDifferent	Dependency	«allDifferent»
owl:oneOf	association type	Dependency	«instanceOf»
owl:AllDiferent	class AllDifferent	Class	«AllDifferent»
owl:distinctMembers	association distinctMembers	Dependency	«distinctMembers»
owl:equivalentProperty	association equivalentProperty	Association	«equivalentProperty»
owl:backwardCompatibleWith	owl.backwardCompatibleWith	Dependency	«backwardCompatibleWith»
owl:imports	owl.imports	Dependency	«imports»
owl:incompatibleWith	owl.incompatibleWith	Dependency	«incompatibleWith»
owl:inverseOf	owl.inverseOf	Dependency	«inverseOf»
owl:priorVersion	owl.priorVersion	Dependency	«priorVersion»