MDA-based Ontology Infrastructure

Dragan Đurić

dragandj@mail.ru
FON – Faculty of Organizational Sciences, University of Belgrade
POB 52, Jove Ilića 154, 11000 Belgrade, Serbia and Montenegro

Abstract. The paper presents Ontology Definition Metamodel (ODM) and Ontology UML Profile 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 and Ontology UML Profile 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 used as a starting point for defining Ontology UML profile that enables using the well-known UML notation in ontological engineering more extensively.

1. Introduction

The Semantic Web and its XML-based languages are the main directions of the future Web development. Domain ontologies [1] 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 [2]. But, UML itself does not satisfy needs for representation of ontology concepts that are borrowed from description logics, 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 the ability to create (using metamodeling) a family of languages [3] 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).

Since Unified Modeling Language (UML) is widely accepted as a modeling language, we define a profile that supports ontology design – Ontology UML Profile. It is a standard extension of UML, and is also based on MOF. Ontology UML Profile is intended to be used as a support to ODM, not as a stand-alone solution for Ontology modeling.

The overview of the Semantic Web languages and OWL is given in the next section, together with the description of the MDA and MOF. In section three 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 four. Section five gives description Of Ontology UML Profile. Based on the paper appendix, we give a 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 the effort of the Good-Old-AI research group (www.goodoldai.org.yu) in developing AIR - a platform for building intelligent information systems.

2. An overview of the Semantic Web, Web Ontology Language, MDA and MOF

The step beyond the World Wide Web is the Semantic Web [4], 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 [1]. 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.

Semantic Web architecture is a functional, non-fixed architecture [6]. Barnes-Lee defined three distinct levels that incrementally introduce expressive primitives: metadata layer, schema layer and logical layer [7]. Languages that support this architecture and the place of OWL are shown in Figure 1.

Common data interoperability in present applications is best achieved by using XML [5]. As shown in the Figure 1, XML supports syntax, while semantics is provided by RDF, RDF Schema and mainly by OWL [8]. In order to provide capabilities for unconstrained representation of the Web knowledge 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 (maximal expressiveness), OWL DL (guaranties computational completeness) and OWL Lite (for starters).

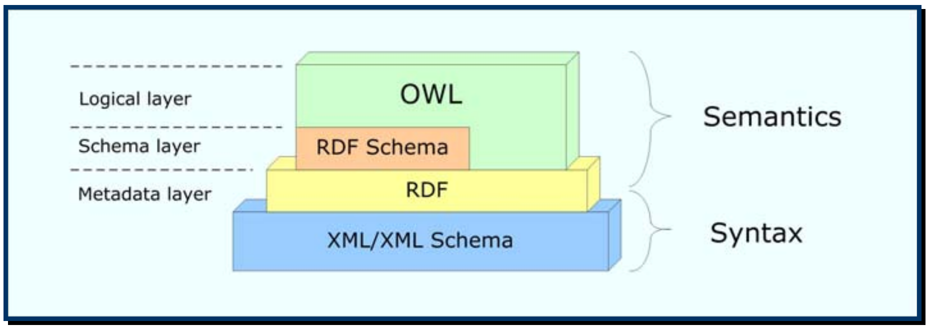


Figure 1. OWL in the Semantic Web architecture

Model Driven Architecture (MDA) [9] 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).

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) [10], Unified Modeling Language (UML) [11] and XML Metadata Interchange (XMI) [12]. Layers are: meta-metamodel (M3) layer, metamodel (M2) layer, model (M1) layer and instance (M0) layer.

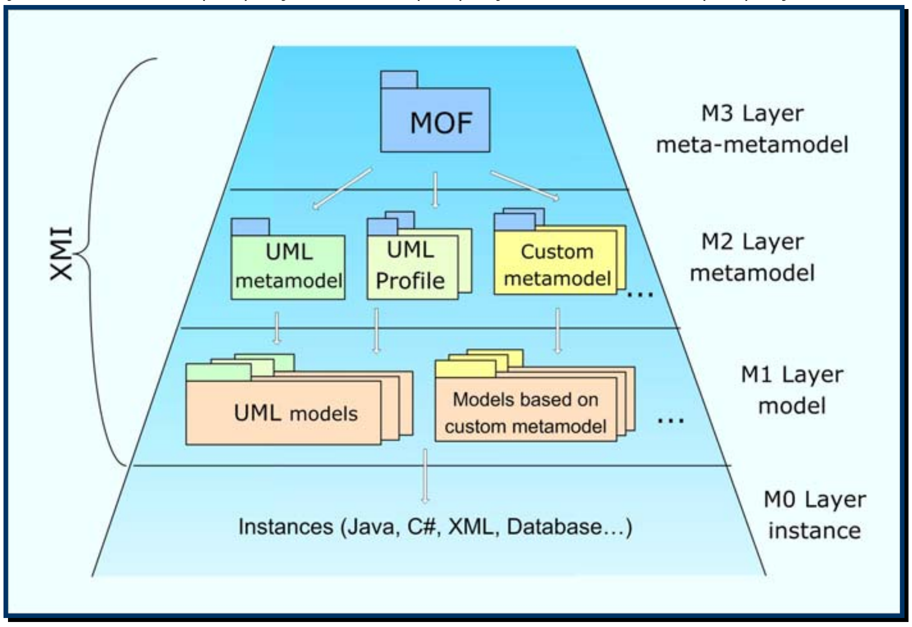


Figure 2. MDA four-layer MOF-based metadata architecture

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 [10]. 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.

3. The Ontology Modeling Architecture

3.1. 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 [2].

MDA and its four-layer architecture provide 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 [14]. 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.

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 [7], 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 [15]. 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.

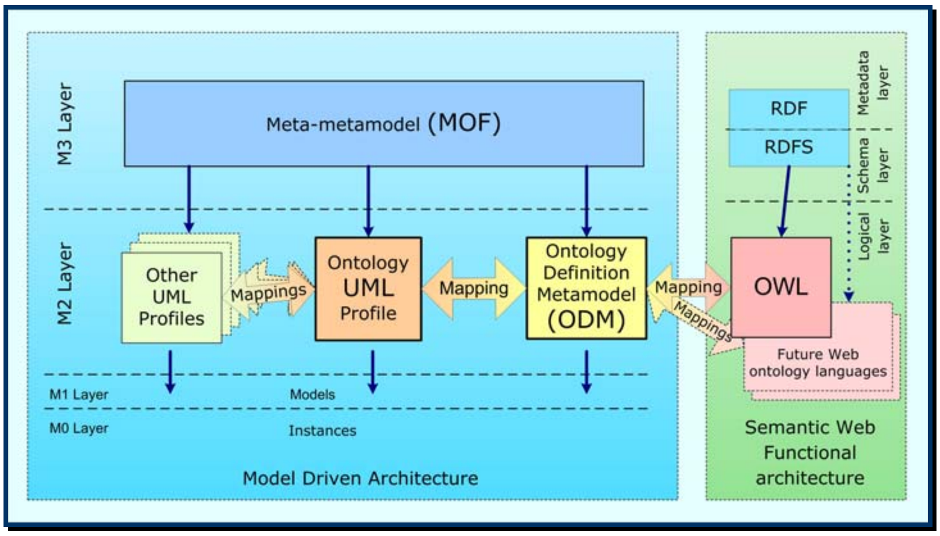


Figure 3. Ontology modeling in the context of MDA and Semantic Web

3.2. 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 [16]. 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.

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), `rdfs:Class` also have function that is similar to a MOF concept of `Classifier`.
`Classifier`	Abstract concept that defines classification. It is specialized by `Class`, `DataType`, etc.		
`Association`	Expresses relationships in the metamodel between pairs of instances of `Classes`	`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`

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 [10]. RDF, RDFS and their concepts are described in detail in W3C documents [6].

4. Essential ODM concepts

4.1. 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.

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`…

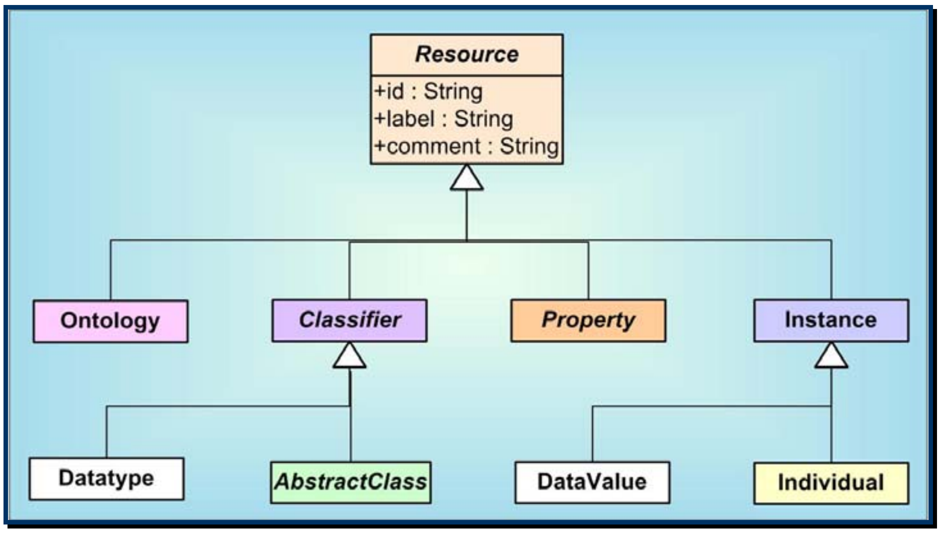


Figure 4. The hierarchy of basic ontology concepts

4.2. 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:

1. A class can be defined by a class identifier (an URI reference) – For example, a Class `Person`.

2. 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`.
3. As a property restriction – Class of all individuals that have the same restriction on some of their characteristics.
4. 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.
5. 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.
6. 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`.
7. `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 [17]. 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`

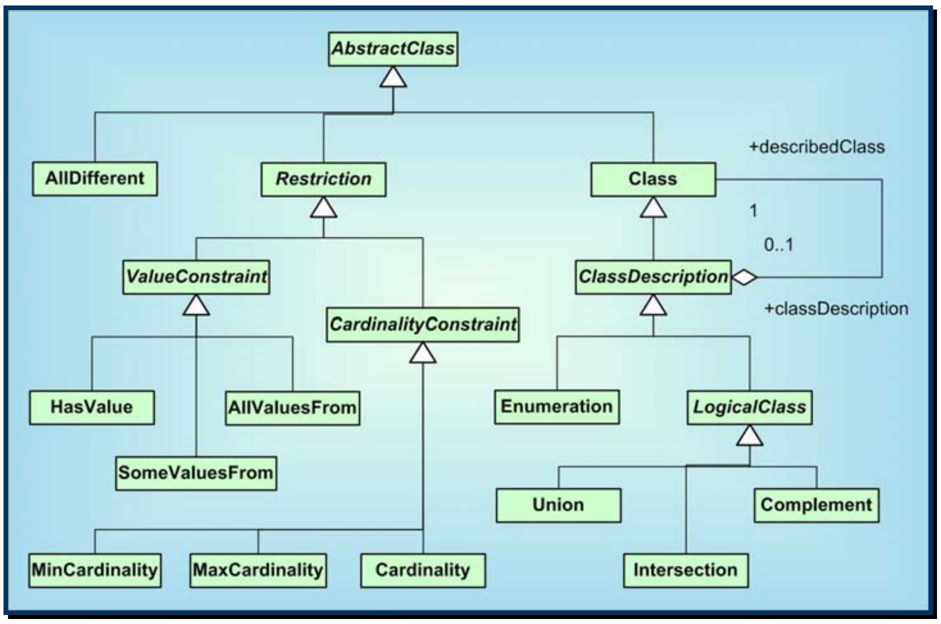


Figure 5. The hierarchy of Ontology Classes in ODM

4.3. 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`.

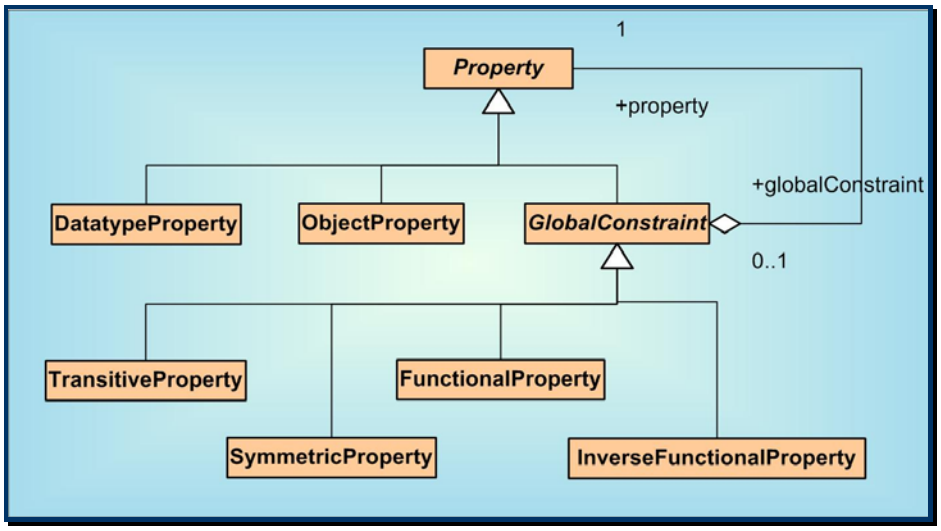


Figure 6. The hierarchy of Ontology Properties in ODM

4.4. 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.

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.

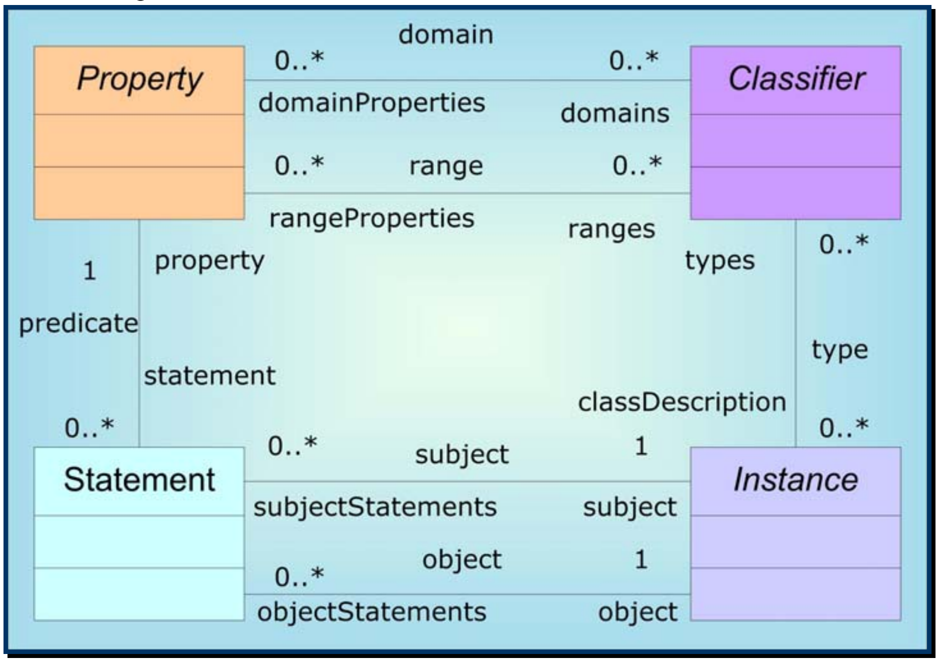


Figure 7. Key relationships among Ontology concepts

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) [10], 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)
```

4.5. 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`.

4.6. Summary of Ontology Definition Metamodel

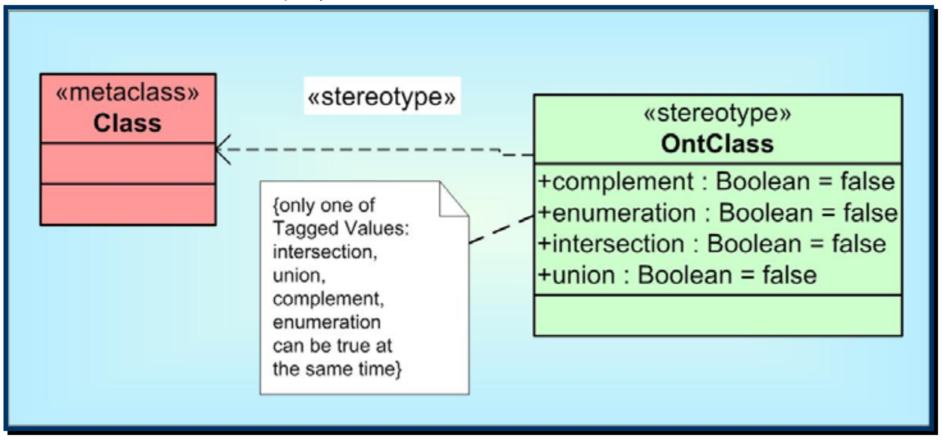
The summary of ODM concepts is given in Table 1 in the Appendix. The first column represents original RDF, RDF(S) and OWL concepts, which are used as the starting point for defining the ODM. The corresponding ODM concepts are listed in second column. The third and fourth columns summarize the Ontology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

5. Ontology UML Profile essentials

UML Profile is a concept used for adapting the basic UML constructs to some specific purpose. Essentially, this means introducing new kinds of modeling elements by extending the basic ones, and adding them to the modeler's tools repertoire. Also, free-form information can be attached to the new modeling elements.

5.1. UML Profile Basics

The basic UML constructs (model elements) can be customized and extended with new semantics by using four UML extension mechanisms defined in the UML Specification [21]: stereotypes, tag definitions, tagged values, and constraints. Stereotypes enable defining virtual subclasses of UML metaclasses, assigning them additional semantics. For example, we may want to define the «OntClass» stereotype, Figure 8, by extending the UML Class metaclass to denote the modeling element used to represent ontologies (and not other kinds of concepts).



«metaclass»
Class
«stereotype»
«stereotype»
OntClass
+complement : Boolean = false
+enumeration : Boolean = false
+intersection : Boolean = false
+union : Boolean = false
{only one of Tagged Values: intersection, union, complement, enumeration can be true at the same time}

Figure 8 – New stereotype definition

Tag definitions can be attached to model elements. They allow for introducing new kinds of properties that model elements may have and are analogous to metaatribute definitions. Each tag definition specifies the actual values of properties of individual model elements, called tagged values. Tag definitions can be attached to a stereotype to define its virtual metaattributes. For example, the «OntClass» stereotype in Figure 8 has a tag definition specifying 4 tagged values (for enumeration, intersection, etc.).

Constraints make possible to additionally refine the semantics of the modeling element they are attached to. They can be attached to each stereotype using OCL (Object Constraint Language) [21] or English language (i.e. spoken language) in order to precisely define the stereotype's semantics (see the example in Figure 8).

More details about UML extension mechanisms can be found in [20] and [21].

A coherent set of extensions of the basic UML model elements, defined for specific purposes or for a specific modeling domain, constitutes a UML profile.

5.2. Design Rationale for Ontology UML Profile

In order to customize UML for modeling ontologies, we define UML Profile for ontology representation, called Ontology UML Profile.

In developing our Ontology UML Profile we used experiences of other UML Profile designers (e.g., see [27]). Applying such experiences to our case, we wanted our Ontology UML Profile to:

- offer stereotypes and tags for all recurring ontology design elements, such as classes, individuals, properties, complements, unions, and the like;
- make specific ontology modeling and design elements easy to represent on UML diagrams produced by standard CASE tools, thus keeping track of ontological information on UML models;
- enable encapsulating ontological knowledge in an easy-to-read format and offer it to software engineers;
- make possible to evaluate ontology UML diagrams and indicate possible inconsistencies;
- support Ontology Definition Metamodel, hence be able to represent all ODM concepts.

Currently, several different approaches to ontology representation in UML have been proposed. We note two major trends among them:

- Extending UML with new constructs to support specific ontology concepts (Property for example) [19].
- Using standard UML and defining a UML Profile for ontology representation [22].

We believe that ontology representation in UML can be achieved without non-standard UML extensions, hence our approach belongs to the latter of the above two trends. In our Ontology UML profile, specific ontology concepts are annotated using the standard UML extension mechanisms described above. Models created with such a UML Profile will be supported by standard UML tools, since they do not add non-standard concepts to UML, thus they are UML models. Since in our approach UML is used to support ODM, not as a stand-alone tool for ontology modeling, Ontology UML Profile will not cover all of the essential ODM (Ontology Definition Metamodel) concepts. Ontology UML Profile should define only constructs for concrete concepts, such as ObjectProperty, Class or Individual, leaving ODM to deal with abstract constructs like Resource, Instance, Classifier, etc, which are not used in development of real ontologies (models), and do not relate to real-world things; they are only introduced to ODM in order to create a coherent hierarchy.

A UML Profile definition in the context of the MDA four-layer metamodeling architecture means extending UML at the metamodel layer (M2). One can understand these extensions as a new language, but also UML as a family of languages [3]. Each of these languages uses UML notation with the four UML extension mechanisms. Recent UML specifications [21] enable using graphical notation for specifying stereotypes and tagged definitions [23]. Thus, all stereotypes and tagged values that are defined in this paper can be shown in this way.

The notation used for stereotype creation of Ontology UML Profile («`OntClass`» stereotype) accomodetes UML's Class («`metaclass`»). Having this graphical notation for the UML extension mechanism can be useful for explaining certain relations between UML constructs and new stereotypes, but also between stereotypes themselves.

Since stereotypes are the principle UML extension mechanism, one might be tempted to think that defining Ontology UML Profile is a matter of specifying a couple of stereoptypes and using them carefully in a coherent manner. In reality, however, it is much more complicated than that. The reason is that there is a number of fine details to take care of, as well as the existence of some conceptual inconsistencies between MDA and UML that may call for alternative design decisions. The following subsections describe the most important Ontology UML Profile concepts in detail. All concepts are summarized in Table 1 in the Appendix.

5.3. Ontology Classes

Class is one of the most fundamental concepts in ODM and Ontology UML Profile. As we noted in the discussion about the essential ODM concepts, there are some differences between traditional UML Class or OO programming language Class concept and ontology class as it is defined in OWL (owl:Class). Fortunately, we are not trying to adopt UML as stand-alone ontology language, since that might require changes to UML basic concepts (Class and other). We only need to customize UML as a support to ODM.

In ODM, Ontology Class concept is represented as an instance of MOF Class, and has several concrete species, according to the class description: Class, Enumeration, Union, Intersection, Complement, Restriction and AllDifferent. These constructs in the Ontology UML Profile are all inherited from the UML concept that is most similar to them, UML Class. But, we must explicitly specify that they are not the same as UML Class, which we can do using UML stereotypes. An example of Classes modeled in Ontology UML Profile is shown in Figure 9.

ODM `Class` identified by a class identifier will have the stereotype `«OntClass»`, `AllDifferent - «AllDifferent»` and `Restriction - «Restriction»`. In ODM, `Enumeration`, `Intersection`, `Union` and `Complement` are descendants of ODM `Class`; in Ontology UML Profile they have stereotypes `«Enumeration»`, `«Intersection»`, `«Union»` and `«Complement»`. The `«OntClass»` stereotype would be extended by each of these new stereotypes. Additionally, enumeration, intersection, union and complement are defined by Boolean tagged values - `enumeration`, `intersection`, `union` and `complement`, which can be added to `«OntClass»` with the constraint that only one of them can be true. This would be similar to the solution used in other UML profiles. A good example is the XML Schema UML profile [24] that has stereotypes for modeling the content model of the XML Schema complex type: any, choice, and sequence. Complex type itself is a distinct stereotype as well. Also, in parallel with these stereotypes, there is a tagged value modelGroup attributed to the complex type stereotype that can take a value from the set consisting of: any, choice, and sequence.

Figure 9 shows various types of ontology classes modeled in UML. The Class Person is an example of an ontology Class that is identified by a class identifier, `TheRollingStones` and `TheWailers` are enumerations, StonesWailersIntersection is an intersection, and StonesWailersUnion is a union. There is a class that represents complement of TheWailers – all individuals that are not members of TheWailers. AllDifferent is an auxiliary class whose members are different individuals. Also shown is an «OntClass» Human and the Dependency «equivalentClass», which means that Person and Human are classes that have the same class description (i.e. all Persons are Humans and vice versa).

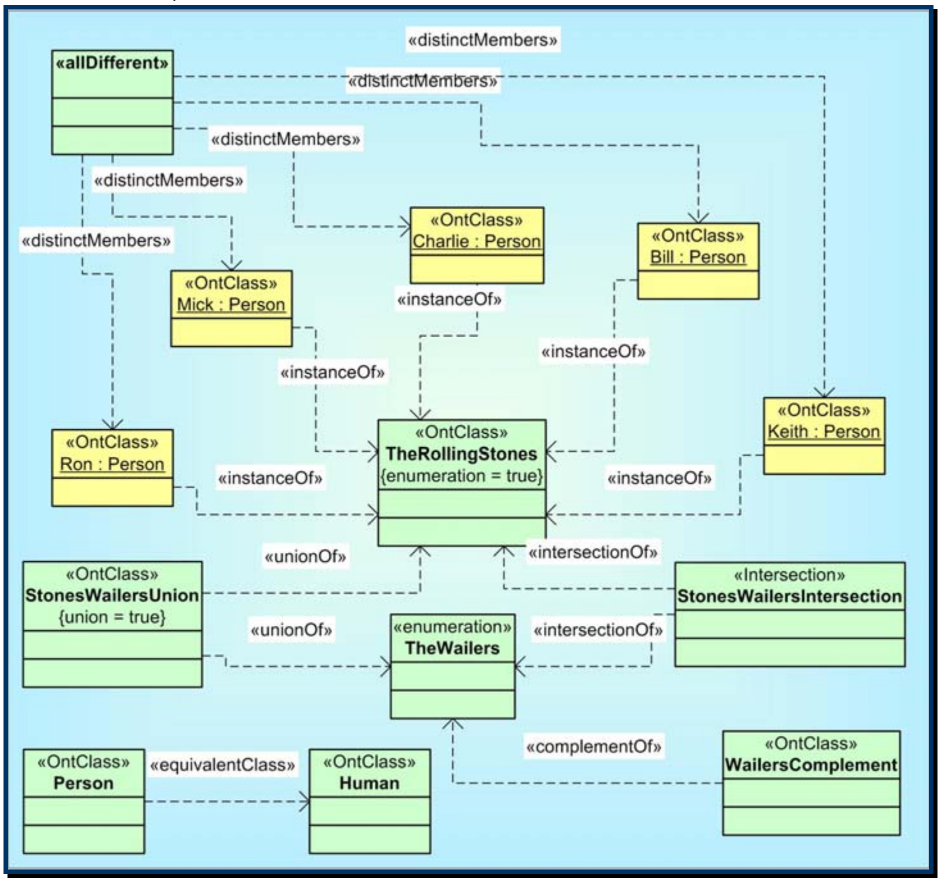


Figure 9 – Class Diagram showing relations between Ontology Classes and Individuals in the Ontology UML Profile

5.4. Individuals

In ODM, an instance of an `AbstractClass` is called Individual. In UML, an instance of a Class is an Object. ODM `Individual` and UML `Object` have some differences, but they are similar enough, so in Ontology UML Profile, Individual is modeled as UML `Object`, which is shown in Figure 9. The stereotype for an object must match the stereotype for its class («`OntClass`» in this case). Stating that some Individual has some type is done in three ways:

1. by using an underlined name of an Individual followed by ":" and its «`ontClass`» name (for example, `Mick:Person` is an `Individual` whose type is `Person`. This is the usual UML method of stating an `Object's` type.
2. by using a UML Dependency's stereotype «`instanceOf`» between an `Individual` and its «`OntClass`». This method is also allowed in standard UML. For example, `Mick` is an instance of `TheRollingStones`.
3. indirectly – through logical operators on «`OntClass`». If some «`OntClass`» is a union, intersection or complement, it is a class of Individuals that are not explicitly defined as its instances. For example, `Mick` is not explicitly defined as a member of `StonesWailersUnion`, but it is its member since he is a member of `TheRollingStones`, which is connected with `StonesWailersUnion` through a «`unionOf`» connection.

Although there are some UML tools (Together, Visio) that allow relations between a UML Class and a UML Object in a UML Class Diagram, many popular UML tools (e.g. Rational Rose, Poseidon for UML) do not support this, even though the UML specification [21] clearly states that Objects and Links can be drawn on Class Diagrams. The authors believe that this is closely related to understanding UML as a graphical notation for modeling and using it with object-oriented programming languages. Another very important issue is related to the MDA metamodeling architecture. UML classes are usually thought of as belonging to the model layer (M1), whereas UML objects are believed to belong exclusively to the instance level (M0). But, this is not quite correct: the UML class and object are defined at the same MDA layer (i.e. M2). Thus, their instances are at the same layer – the model layer (i.e. M1). Actually, a UML object models a thing from the real world [26]. But, objects only model real world things; they are not real things (e.g. in Figure 9 the object Mick only models an instance of Human). Then, how can we distinguish between the instance-of relation between objects and classes, and, on the other hand, between UML Class (metaclass) and some concrete class? We believe that Atkinson and Kühne [25] have adequately proposed the solution to this problem by introducing two kinds of instance-of relations: linguistic and ontological. The linguistic instance-of relation is the instance-of relation between concepts from different layers (UML Class definition and some concrete class, for instance `TheWailers`). The ontological instance-of relation is the instance-of relation between concepts that are at the same linguistic layer, but which are at different ontological layers (for instance, «`OntClass`» `Person` and object `Keith` are at different ontological layers since Human is the class (type) of `Keith`).

5.5. Ontology Properties

Property is one of the most unsuitable ontology concepts to model with object-oriented languages and UML. The problem arises from the major difference between Property and its similar UML concepts – Association and Attribute. Since Property is an independent, stand-alone concept, it can not be directly modeled with Association or Attribute, which can not exist on their own. Some authors [19] suggested extending UML with new constructs to support the stand-alone Property, introducing aspect-oriented programming concepts into UML. In our view, this solution is rather extreme, since it demands non-standard changes to UML. We try to introduce Property in UML in some other way instead.

Since Property is a stand-alone concept it can be modeled using a stand-alone concept from UML. That concept could be the UML Class' stereotype «`Property`». However, Property must be able to represent relationships between `Resources` (`Classes`, `Datatypes`, etc. in the case of UML), which the UML Class alone is not able to do. If we look at the ODM Property definition more closely, we will see that it accomplishes relation representation through its range and domain. According to the ODM Model, we found that in the Ontology UML Profile, the representation of relations should be modeled with UML Association's or UML Attribute's stereotypes «`domain`» and «`range`». In order to increase the readability of diagrams, the «`range`» association is unidirectional (from a `Property` to a `Class`).

ODM defines two types (subclasses) of Property – `ObjectProperty` and `DatatypeProperty`. `ObjectProperty`, which can have only `Individuals` in its `range` and `domain`, is represented in Ontology UML Profile as the Class' stereotype «`ObjectProperty`». `DatatypeProperty` is modeled with the `Class'` stereotype «`DatatypeProperty`».

An example of a Class Diagram that shows ontology properties modeled in UML is shown in Figure 10. It contains four properties: two «`DatatypeProperty`»s (name and `socialSecurityNumber`) and two «`ObjectProperty`»s (`nationality and colleague`) UML Classes. In cooperation with «`domain`» and «`range`» UML Associations, or «`domain`» and «`range`» UML Attributes, they are used to model relationships between «`OntClass`» UML Classes. Tagged values describe additional characteristics, for example, «`ObjectProperty`» colleague is symmetric (if one Person is a colleague of another Person, the other Person is also a colleague of the first Person) and transitive (if the first Person is a colleague of the second Person, who is a colleague of the third Person, the first and third Person are colleagues). In ODM, these characteristics are added to an ODM Class applying the Decorator Design Pattern [17]. The transformation that maps an Ontology UML Profile model to an ODM model should create one decoration of an ODM Property per attribute of Ontology UML Profile «`ObjectProperty`» or «`DatatypeProperty`».

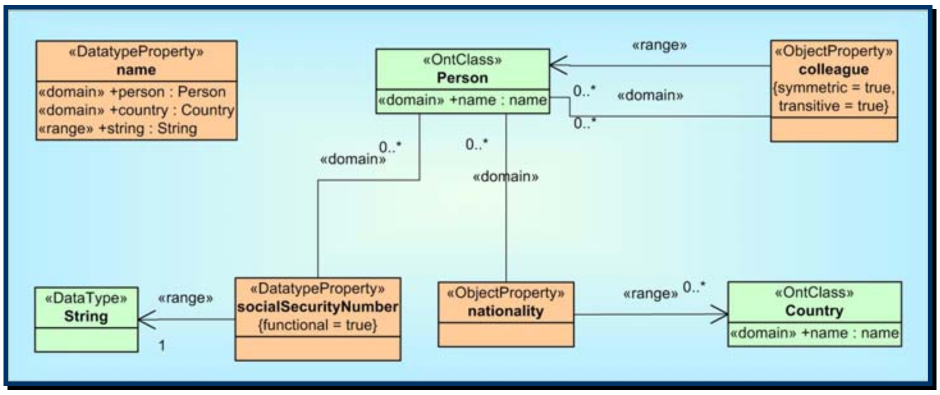


Figure 10 – Ontology Properties shown in UML Class Diagram

There is an important issue that must be clarified with this diagram. In UML, relations are represented by Associations (graphically represented as lines) or Attributes, which looks nice and simple. Ontology UML Profile diagrams may look overcrowded, since each relationship requires a box and two lines to be properly represented. The solution shown in this paper uses standard graphical symbols, but UML allows custom graphical symbols for a UML Profile. For example, a custom graphical symbol for Property could be a tiny circle with lines, which reduces the space on diagrams. Also, additional custom settings, like distinct colors for «`OntClass`» (green), «`ObjectProperty`» (orange) or «`DatatypeProperty`» (orange) in this paper, can be used to increase the diagram readability. For the sake of readability, this UML Profile allows two styles of «`DatatypeProperty`» domain and range presentation. An example of the first style (a UML Class with two UML Associations) is `socialSecurityNumber`, and an example of the second one (a Class with Attributes as domain or range) is name. The second style is allowed only for «`DatatypeProperty`» whose range multiplicity is equal or less than one. So, if a «`DatatypeProperty`» has range multiplicity of 0..1 or 1, the style with Attributes can be used to reduce the clutter.

5.6. Statement

ODM Statement is a concept that represents concrete links between ODM instances – `Individuals` and `DataValues`. In UML, this is done through Link (an instance of an Association) or AttributeLink (an instance of an

Attribute). Statement is some kind of instance of a Property, which is represented by the UML Class' stereotype («`ObjectProperty`» or «`DatatypeProperty`»). Since in UML a Class' instance is an Object, in Ontology UML Profile Statement is modeled with Object's stereotype «`ObjectProperty`» or «`DatatypeProperty`» (stereotype for Object in UML must match the stereotype for its Class' stereotype). UML Links are used to represent the subject and the object of a Statement. To indicate that a Link is the subject of a Statement, LinkEnd's stereotype «`subject`» is used, while the object of the Statement is indicated with LinkEnd's stereotype «object». LinkEnd's stereotype is used because in UML Link can not have a stereotype. These Links are actually instances of Property's «`domain`» and «`range`». In brief, in Ontology UML Profile Statement is represented as an Object with two Links – the subject Link and the object Link, which is shown in Figure 11. The represented Persons Mick and Keith are colleagues. They both have UK (United Kingdom) nationality.

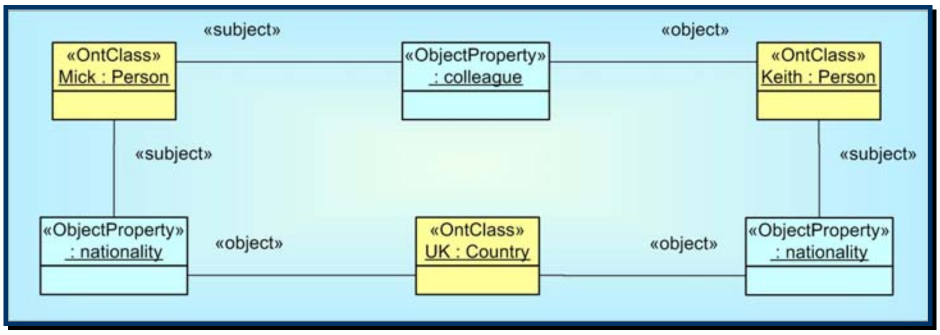

«subject»
«object»
«OntClass» Mick : Person
«ObjectProperty» : colleague
«OntClass» Keith : Person
«subject»
«subject»
«ObjectProperty» : nationality
«object»
«OntClass» UK : Country
«object»
«ObjectProperty» : nationality

Figure 11 – Individuals and Statements shown in a UML Object Diagram.

As with Ontology Properties, the diagram's readability can be further increased by using distinct colors and custom graphical symbols. A tiny circle can be used instead of the standard box for representing a Statement in order to reduce clutter on a diagram.

5.7. Summary of Ontology UML Profile

We have seen in detail how the most important Ontology Definition Metamodel concepts are translated into Ontology UML Profile. Table 1 in the Appendix shows the summary of all Ontology UML Profile concepts, together with the corresponding ODM and OWL concepts.

6. Conclusions

The Ontology Definition Metamodel and Ontology UML Profile defined in this paper are 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 UML and ODM are defined as MOF-compliant languages it is possible to store ontologies in MOF-based repositories, to store ontology diagrams in a standard way (UML2 XMI), as well as to share and interchange ontologies using XMI.

The proposed ODM and Ontology UML Profile can be considered as a part of the effort to specify standard ontology metamodel. Their important feature is that they are based on OWL. With the Ontology UML Profile, the ODM concepts can be used as stereotypes in the UML models (similar to UML CORBA Profile or other OMG's UML Profiles).

The possibilities of defining other AI metamodels in MOF should and will be explored in the future work. This means that MDA and MOF will be the integrating point for metamodels, both common and AI-related. Further plans include using Java Metadata Interface (JMI) [18] to enable creation, storage, access, discovery, and exchange of ODM-defined ontologies using standard Java interfaces.

7. References

1. Gruber, T. R., "A translation approach to portable ontology specifications", Knowledge Acquisition, Vol. 5, No. 2, 1993.
2. 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.
3. Duddy, K., "UML2 Must Enable A Family of Languages", Communications of the ACM, Vol. 45, No. 11, November 2002, pp 73-75.
4. Tim Berners-Lee, Weaving the Web, Orion Business Books, London, 1999.
5. 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.
6. 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.
7. Tim Berners-Lee, "Semantic Web Road Map", W3C Design Issues, http://www.w3.org/DesignIssues/Semantic.html, 1998.
8. van Harmelen, F., et al, "OWL Web Ontology Language Reference", W3C Working Draft, http://www.w3.org/TR/2003/WD-owl-ref-20030331/, 2003.
9. 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.

10. “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.

11. Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language User Guide, Addison-Wesley, Massachusetts, 1998.

12. “OMG XMI Specification, v1.2”, OMGDocument formal/02-01-01, http://www.omg.org/cgi-bin/doc?formal/2002-01-01, 2002.

13. 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.

14. “Ontology Definition Metamodel Request for Proposal”, OMG Document: ad/2003-03-40, http://www.omg.org/cgi-bin/doc?ad/2003-03-40, 2003.

15. 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.

16. 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.

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

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

19. Baclawski, K., et al., “Extending UML to support ontology engineering for the Semantic Web”. Fourth International Conference on The Unified Modeling Language, volume 2185, pages 342–360. Springer-Verlag, Berlin, October 2001.

20. Rumbaugh, J., Jacobson, I., Booch, G., “The Unified Modeling Language Reference Manual”, Addison-Wesley, 1998.

21. “OMG Unified Modeling Language Specification”, Object Management Group, http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.zip, March 2003.

22. Baclawski, K. et al, “UOL: Unified Ontology Language”, Assorted papers discussed at the DC Ontology SIG meeting, http://www.omg.org/cgi-bin/doc?ontology/2002-11-02, 2002.

23. Kobryn, C., “The Road to UML 2.0: Fast track or Detour”, Software Development Magazine, April 2001, http://www.sdmagazine.com/documents/s=732/sdm0104b/0104b.htm.

24. Carlson, D., Modeling XML Applications whit UML: Practical E-Business Applications, Addison-Wesley, Boston, USA, 2001.

25. Colin Atkinson, Thomas Kühne, , “Model-Driven Development: A Metamodeling Foundation” (Spec. issue on Model-Driven Development), IEEE Software, Vol. 20, No. 5, Sep/Oct, 2003, pp 36-41.

26. Colin Atkinson, Thomas Kühne, "Rearchitecting the UML Infrastructure", ACM Transactions on Modeling and Computer Simulation, Vol. 12, No. 4, October 2002, pp 290–321

27. Juerjens, J., Secure Systems Development with UML. Springer-Verlag, Berlin, 2003.

Appendix

Table 1. ODM and Ontology UML Profile Summary

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.) andv «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:InverseFunctiona Property	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	Dependency	«equivalentProperty »
owl:backwardCompaibleW ith	owl.backwardCompatibleW ith	Dependency	«backwardCompatib le With»
owl:imports	owl.imports	Dependency	«imports»
owl:incompatibleWith	owl.incompatibleWith	Dependency	«incompatibleWith»
owl:inverseOf	owl.inverseOf	Dependency	«inverseOf»
owl:priorVersion	owl.priorVersion	Dependency	«priorVersion»

Dragan Djuric is a MSc student at FON – Faculty of Organizational Sciences, 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.