Journal of Web Engineering, Vol. 4, No.2 (2005) 118-143

BRIDGING MDA AND OWL ONTOLOGIES

DRAGAN V. GAŠEVIĆ
Laboratory for Ontological Research, School of Interactive Arts and Technology,
Simon Fraser University, Surrey, Canada
dgasevic@acm.org

DRAGAN O. DJURIĆ, VLADAN B. DEVEDŽIĆ
GOOD OLD AI Research Group, FON – School of Business Administration,
University of Belgrade, Serbia and Montenegro
dragandj@gmail.com, devedzic@fon.bg.ac.yu

Received July 16, 2004
Revised March 3, 2005

Web Ontology Language (OWL) and Model-Driven Architectures (MDA) are two technologies being developed in parallel, but by different communities. They have common points and issues and can be brought closer together. Many authors have so far stressed this problem and have proposed several solutions. The result of these efforts is the recent OMG's initiative for defining an ontology development platform. However, the problem of transformation between ontology and MDA-based languages has been solved using rather partial and ad hoc solutions, most often by XSLT. In this paper we analyze OWL and MDA-compliant languages as separate technological spaces. In order to achieve a synergy between these technological spaces we define ontology languages in terms of MDA standards, recognize relations between OWL and MDA-based ontology languages, and propose mapping techniques. In order to illustrate the approach, we use an MDA-defined ontology architecture that includes ontology metamodel and ontology UML Profile. Based on this approach, we have implemented a transformation of the ontology UML Profile into OWL representation.

Key words: Ontology development, MDA, UML Profiles, OWL, model transformations
Communicated by: D Schwabe & K Turowski

1 Introduction

The Semantic Web initiative tries to establish better semantic connections between different resources on the Web using AI techniques. Domain ontologies are the most prominent part of this research that should provide a formal way to represent a conceptualization of some domain [30]. Accordingly, many ontological languages are defined within the Semantic Web community. Most of these languages are XML-based (e.g. SHOE, OML, RDF Schema - RDFS, DAML, DAML+OIL, etc.) [29]. Even though Semantic Web languages use XML, they have more rigorous foundation closely related to the well-known AI paradigms (e.g. Description Logic, semantic networks, frames, etc.). Thus, most of current Semantic Web ontologies are developed in AI laboratories.

Researchers have been trying to integrate the ongoing software engineering efforts with the concept of the Semantic Web for a while [35]. The main question they want to answer is how to develop the Semantic Web ontologies using well-accepted software engineering languages and techniques in order to have a large number of practitioners developing and using ontologies in real-world applications. Many researchers have previously suggested using UML in order to solve this problem. However, UML is based on object-oriented paradigm, and has some limitation regarding

ontology development (e.g. properties in ontology languages are fist-class concepts, while UML properties (i.e. attributes and associations) are defined in the scope of a class they belong to [3]. Furthermore, UML classes and their inheritance covers behavioral characteristics of abstractions they model whereas ontologies considers model-theoretical ones. For a detail overview of differences between UML and ontology language see [3]). Hence, we can only use UML in initial phases of ontology development. We believe that these limitations can be overcome using UML extensions (i.e. UML profiles) [22], as well as other Object Modeling Group (OMG) standards, like *Model Driven Architecture – MDA*. In addition, if we want to offer a solution consistent with MDA proposals, we should also support automatic generation of completely operational ontology definitions (e.g. in OWL language) that are model driven [52]. Currently, the most important direction toward this goal is the one pursued by a dedicated research group within OMG that tries to converge many different proposals of solutions to this problem [46]. The result of this effort should be a standard language (i.e. a metamodel) based on the MDA standards [39] and the W3C *Web Ontology Language* (*OWL*) recommendation [5].

Technological spaces have been recently introduced as a means to figure out how to work more efficiently by using the best possibilities of different technologies [37]. In this paper we use this concept to bring closer together two technological spaces: MDA-compliant languages and OWL. To this end, we identify similarities between these technological spaces regarding their epistemological organization and layered architecture. For example, MDA has a four-layer metamodeling architecture whereas the ontology languages like OWL have three-layer architecture according to [16]. Also, the XML technological space is important for our analysis since both MDA and OWL use XML formats for sharing metadata. In order to develop valid transformations we find equivalences among them in terms of epistemological equivalencies between concepts existing in all of them. As a result we give recommendations on how to develop transformations between the MDA languages and OWL as well as what specific technologies can be used for implementation. On top of those theoretical considerations we developed a practical example. The example implements an XSLT-based solution for transforming an ontology UML profile into OWL language. That way, we illustrate all pros and cons of such an approach to bridging gaps between UML Profiles (i.e. MDA TS) and OWL TS.

The next section formally defines MDA, metamodeling, UML Profiles, and technological spaces. Section 3 briefly depicts an example of an MDA-based ontology development architecture, which we defined according to the OMG initiative. Using this architecture, we give a conceptual solution for mapping between MDA-compliant ontology languages and OWL in section 4. Section 5 contains an XSLT-based implementation example for transforming ontology UML Profile into OWL, as well as our experiences in using this transformation. In Section 6 we give an overview of related work in transforming MDA-based models into ontology languages, whereas in section 7 we discuss some further specifics of the proposed solution. The approach presented in the paper is a part of research efforts of the GOOD OLD AI research group (http://goodoldai.org.yu). The efforts are focused on development of an ontology metamodeling architecture based on the current OMG initiative for Ontology Definition Metamodel (ODM) [21].

2 Formal framework

In this section we describe the MDA-supported standards, give important definitions related to these standards, and define technological spaces. We rely on these definitions in the rest of the paper. We need all these concepts in order to explain metamodeling described ontology languages as well as their transformations to OWL ontologies.

2.1 MDA basics

Our work is based on MDA – an ongoing software engineering effort under the auspices of OMG. The central part of MDA is the four-layer architecture that has a number of standards defined at each of its layers (see Figure 1). Most of MDA standards are developed as metamodels using metamodeling. The topmost layer (M3) is called meta-metamodel and the OMG's standard defined at this layer is Meta-Object Facility (MOF) [45]. This is a self-defined language intended for defining metamodels. In terms of MDA, a metamodel makes statements about what can be expressed in the valid models of a certain modeling language. In fact, a metamodel is a model of a modeling language [51]. Examples of the MDA's metamodels are UML and Common Warehouse Metamodel (CWM). The MDA's metamodel layer is usually denoted as M2. At this layer we can define a new metamodel (e.g., a modeling language) that would cover some specific application domains (e.g., ontology development). The next layer is the model layer (M1) – the layer where we develop real-world models (or domain models). In terms of UML models, that means creating classes, their relations, states, etc. There is an XML-based standard for sharing metadata that can be used for all of the MDA's layers. This standard is called XML Metadata Interchange (XMI) [50]. The bottom layer is the instance layer (M0). There are two different approaches to explaining this layer, and we note both of them:

1. The instance layer contains instances of the concepts defined at the model layer (M1), e.g. objects in programming languages.
2. The instance layer contains things from our reality – concrete (e.g. Mark is an instance of the Person class, Lassie is an instance of the Dog class, etc.) and abstract (e.g. UML classes – Dog, Person, etc.) [2]. Other authors also mentioned that difference, like Bézivin [9], who says that the M0 layer covers program executions as well. Accordingly, the ongoing UML 2.0 [49] accepted this approach to the M1 layer, so in the reset of the paper we rely on such a presumption.

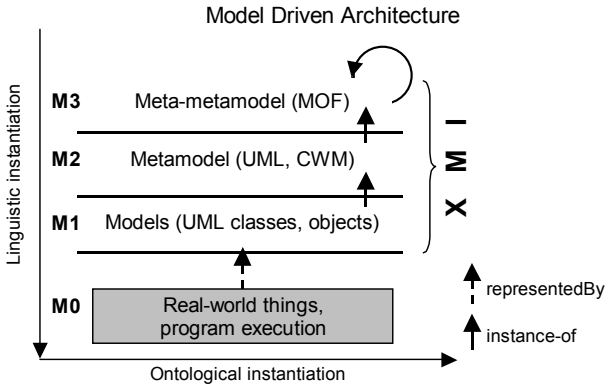

Figure 1 The four-layer Model Driven Architecture and its orthogonal instance-of relations: linguistics and ontological

In UML, both classes and objects are at the same layer (the model layer) in the MDA four-layer architecture. Actually, MDA layers are called linguistic layers, and there is linguistic instatiation relation between them. An exception is the relation between M1 and M0 layers. Since we create models of the reality (i.e. the M0 layer) as well as executable programs at the M1 layer. That means, the M1 layer represents the M0 layer. On the other hand, concepts from the same linguistic layer can be at different ontological layers. Hence, UML classes and objects are at different ontological layers, but at the same linguistic layer. There is no any explicit definition how many ontological layers can be at a linguistic layer. We give an excerpt from the Petri net ontology [25] as an illustration of this

approach (see Figure 2). There is the concept of modules in Petri nets that means components of Petri net models we can reuse in other different Petri net models. In fact, Petri net modules are similar to UML classes – it is a sort of template that we use for creating module instances. However, both of them we modeled using UML classes, but there was a need to make difference between them. So, we modeled modules as a metaclass, while module instances as a regular class. Note that we did not change UML metamodel by defining Module metaclass, but we just created ontological metaclass. That means, modules are at the O2 ontological layer, while module instances are at the O1 ontological layer. Finally, module instances represent (model) real Petri net modules. As Petri nets themselves are modeling tool, they are at the O0 ontological layer, but sill at the M1 (i.e. model) linguistic layer.

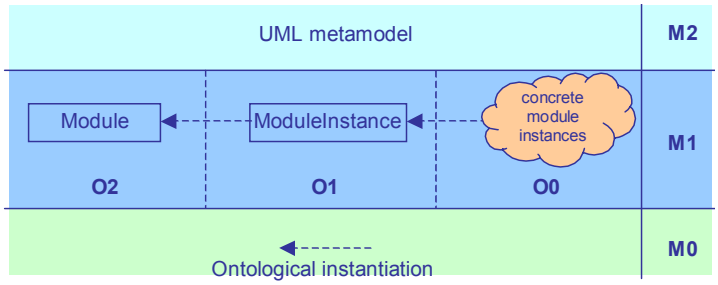

Figure 2 An illustration of ontological layers: An excerpt of the Petri net ontology consisting of three ontological layers (O2, 01, O0) at the same linguistic layer

2.2 Specific MDA metamodels and UML Profiles

One possible solution for using MDA capacities in a specific domain is to develop a metamodel that would be able to model relevant domain concepts. That means creating a domain language (i.e. the metamodel) using metamodeling; such a language is created using MOF. Having defined a domain specific metamodel, we should develop suitable tools for using that metamodel. However, it is rather expensive and time-consuming to develop new tools (e.g. for ontology development compliant with an MDA-based language), so we try to use existing, well-known tools. Current software tools do not implement many of the MDA basic concepts. However, most of these tools are currently oriented primarily towards UML and the M1 layer (i.e., the model layer) [26]. Generally, UML itself is a MOF-defined general-purpose language (i.e. a metamodel) that contains a set of core primitives. The problem of tools can be overcome using UML Profiles – a way to adapt UML to specific domains (e.g. ontology development). UML Profiles extend the UML metamodel with domain-specific primitives (through stereotypes, tagged values, and constraints), and hence these primitives can be used as the regular UML concepts. With Profiles, UML can be seen as a family of languages [22].

A very important issue is the place of UML Profiles in the MDA four-layer architecture. The UML specification states that UML Profiles are defined at the metamodel layer (M2), and thus they are meta-concepts. Here we use a definition of UML Profiles in a strict metamodeling framework [1], where UML Profiles are placed at both the metamodel layer (M2) and the model layer (M1).

2.3 Technological spaces

Nowadays, using only a single technology in solving different engineering problems is usually not enough. For example, software engineers can benefit from ontological engineering, or database developers can find useful improvements in using the XML technology. Problems of bridging different technologies are discussed in [37] where the term of technological spaces is introduced. A

technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Although some technological spaces are difficult to define, they can be easily recognized (e.g. XML, MDA, and ontology technological spaces in the case of approaching MDA and OWL). In order to get a synergy of different technological spaces we should create bridges between them, and some of these bridges are bi-directional. The bridges can be created in a number of ways (e.g. in the XML technological space by using XSLT, in ontological engineering through transformations that can be mapped into XSLT, etc.). Of course, sometimes it is not enough to create a bridge between technological spaces using just one technology (e.g. XSLT), but we should/could employ two-three of them (e.g., for importing/exporting between MDA and XML we can use both XSLT and QVT). Note that technological spaces can be classified according to their layers of abstraction (e.g. MDA and ontological engineering are high-level spaces, whereas XML and databases are low-level spaces). The Semantic Web integrates XML and ontological engineering technological spaces.

Currently, there is an OMG initiative (Request for Proposal – RFP) entitled MOF 2.0 Query/View/Transformation (QVT) [47]. This is a platform-neutral part of MDA aiming to define a language for querying and transforming models as well as viewing metamodels. Atlas Transformation Language (ATL) is an example of submission to the OMG's QVT RFP. ATL can be used to bridge different technological spaces, and the potentials of this language are shown in [7] where it is used to transform XSLT documents to XQuery. Although this transformation can be done inside the XML technological space through XSLT (because an XSLT document is a valid XML document), it is performed in both MDA and XML technological spaces.

3 Ontology development and MDA

In this section we first explain an OMG effort for building a set of standards supporting ontology development. First, we describe starting points of OMG's proposal for Ontology Definition Metamodel. Then, we give a short overview of the ontology metamodeling architecture [21] we developed on top of OMG's initiative. Finally, we describe the Ontology UML Profile [21] since we use that UML Profile to demonstrate a practical implementation of the mapping an MDA-compatible language into OWL (see Section 5).

3.1 Starting points for OMG's ontology standard

All concepts we explained in the previous section are developed to be used in software engineering tools and by software engineering practitioners. On the other hand, the practitioners are manly unfamiliar with ontology development techniques. Researches proposed using well-known tools and techniques (e.g. UML) for ontology development as a solution of this problem [35]. The problem of using UML for ontology development has been firstly addressed in [14]. In fact, this was a pioneering work in integrating MDA and ontologies. Until now there were another few attempts to use MDA standards for the benefit ontological engineering. Currently, there is an OMG RFP aiming to define a suitable language for modeling Semantic Web ontology languages in the context of MDA [39]. In the context of that RFP we present in Figure 3 our proposal for such an architecture [21]. The key components in the architecture are:

- Ontology Definition Metamodel (ODM);
- Ontology UML Profile (OUP) – a UML Profile that supports UML notation for ontology definition;

- Two-way mappings between OWL and ODM, ODM and OUP, and from OUP to other UML profiles.

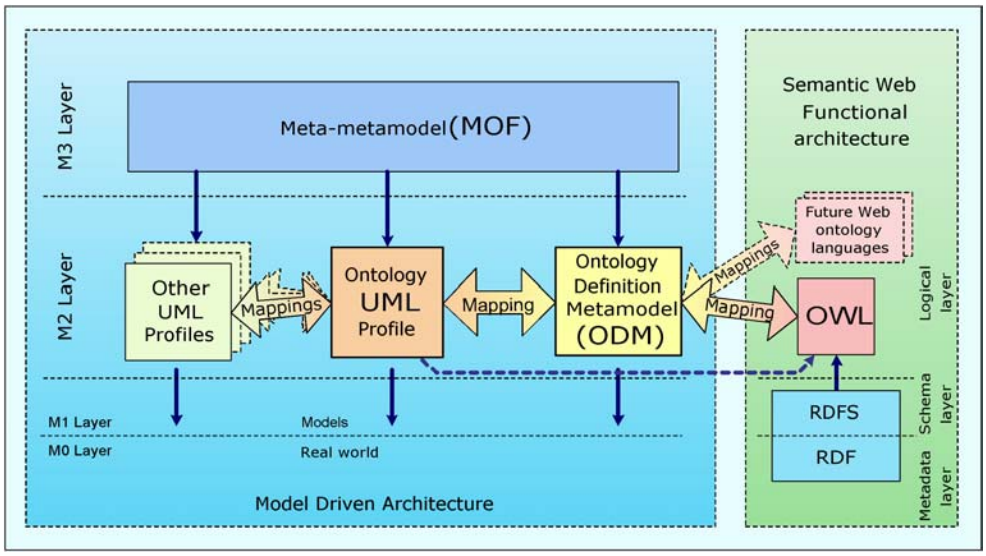

Figure 3 Ontology modeling in the context of MDA and the Semantic Web (This figure does not try to identify the relations between MDA and OWL layers)

In Figure 3 we inserted mappings between OUP and OWL although it is not a part of the ongoing OMG specification. The transformation from OUP to OWL is a practical extension of present UML tools that gives them capability to be used for full development of ontology described by a real Semantic Web language. We note this transformation, as the example we present in section 5 is its implementation. Further discussion about the transformation is given in section 7.

3.2 Ontology metamodeling architecture

A natural way to define an MDA-based ontology language is to develop a metamodel. In terms of OMG RFP this metamodel is entitled ODM. ODM should be designed to enclose common ontology concepts. A good starting point for ODM construction is OWL since it is the result of evolution of existing ontology representation languages, and is a W3C's recommendation [5]. It is at the Logical layer of the Semantic Web [6], on top of RDF Schema (Schema layer). In order to make use of graphical modeling capabilities of UML, it would be useful for ODM to have a corresponding UML Profile [53]. This profile enables graphical editing of ontologies using UML diagrams as well as other benefits resulting from the use of the mature UML CASE tools. Both UML and ODM models are serialized in XMI format so the two-way transformation between them can be done using XSLT. OWL can be also represented in XML format, so another pair of XSLTs should be provided for two-way mapping between ODM and OWL. For mapping from OUP into other, technology-specific UML Profiles, additional transformations can be added to support using ontologies in application design in other domains and vice versa.

We have defined ODM using MOF [19] [21]. A brief comparative description of the most important metamodeling constructs in MOF and RDF(S), is shown in Table 1. Using that comparative analysis we defined ODM, and its comprehensive overview can be seen in [21]. Detailed description

of MOF can be found in the OMG's MOF specification document [45]. RDF, RDFS and their concepts are described in detail in W3C documents [12]. The ODM gives us a metamodel-based semantic foundation [51] for ontology languages, so we can use MDA capabilities for ontology development. But, if we want to use standard CASE tools for ontology development we need a UML Profile whose formal semantics is compliant with the ODM. Thus, in the next subsection we shortly outline the ODM-based UML Profile – OUP.

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 a 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`

3.3 Ontology UML Profile

Here, we briefly outline OUP defined as a part of the ODM-compatible ontology metamodeling architecture we developed [21]. OUP's details are given in [20]. In fact, OUP uses the standard UML extension and customization mechanisms defined in the UML Specification [49]: stereotypes, tag definitions, tagged values, and constraints. *Stereotypes* enable defining virtual subclasses of UML metaclasses, assigning them additional semantics. Creating UML stereotypes means extending concepts that are regular part of the UML metamodel (e.g. class, association, dependency). Since we extend UML metamodel, those extensions are done at M2 layer. In development of our Ontology UML Profile we used experiences of other UML Profile designers (e.g., see [31]). Applying those experiences to our case, we wanted our OUP 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 in UML diagrams produced by standard CASE tools, thus keeping track of ontology information in UML models;
- enable encapsulating ontological knowledge in an easy-to-read format and offer it to software engineers;
- make possible evaluation of ontology UML diagrams that would indicate possible inconsistencies;
- support ODM, hence be able to represent all ODM concepts.

In the rest of this subsection we try to illustrate some of OUP stereotypes in order to give a reader a better picture of the UML Profile under study. All of those stereotypes are introduced at the M2 layer by using the UML2 *extension* relation for defining stereotypes. The examples are parts of the well-known Wine ontology and all of them belong to the M1 layer.

Since 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`) we used stereotyped <<OntClass>> UML classes to model ontology classes in OUP. However, ontology languages (i.e. ODM and OWL) have several other class species, namely `Enumeration`, `Union`, `Intersection`, `Complement`, `Restriction` and `AllDifferent`. These constructs in OUP are all inherited from the UML concept that is the most similar to them, i.e. from UML `Class`. That means, we defined a new class stereotype for each of them.

In Figure 4 we show a part of the well-known Wine ontology. `WineDescriptor` is a class equivalent to the union of classes `WineTaste` and `WineColor`, whereas the `WineColor` class is an enumeration of the `WineColor` class' instances (i.e. individuals): `White`, `Rose`, and `Red`. Note that there are two anonymous classes (`Union` and `Enumeration`) in Figure 4. Those classes are defined through other classes (e.g. the anonymous `Enumeration` is defined in the class `WineColor`) and cannot be used outside of their definitions. Note also that we use the tag value *odm.anonymous* with the value *true*, to denote anonymous classes. This helps us differentiate between anonymous and non-anonymous classes in automatic transformation of OUP models.

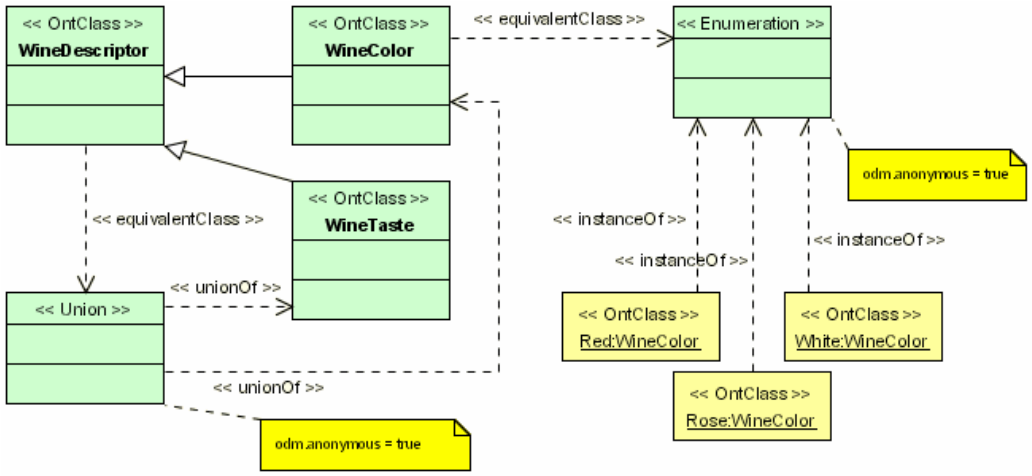

Figure 4 Ontology UML Profile class-oriented stereotypes (an excerpt of the Wine ontology)

In UML, an instance of a `Class` is an `Object`. Since ontology (i.e. ODM and OWL) individuals and UML `Object` have some differences, OUP `Individuals` are modeled as stereotyped UML

Objects, as it is shown in Figure 4. Here we had difficulties deciding on what stereotype to attach to UML objects to make them represent ODM individuals. It would be natural to have a stereotype with the name «Individual», but the UML specification [48] explicitly prompts that the stereotype for an object must match the stereotype for its class. Accordingly, in OUP we have attached the «OntClass» stereotype to the OUP instances.

The next OUP concept we discuss is property. In ontology languages (e.g. OWL) Property is a stand-alone concept and can be modeled using a stand-alone concept in UML. This is opposite to UML class Attributes that are part of UML classes (i.e. each UML attribute is owned by an UML class). That concept could be the UML Class' stereotype «Property». However, Property must be able to represent relationships between Resources (Classes, Datatypes, etc. in case of UML), which UML Class alone is not able to do. In fact, in UML it is the role of relationships. Furthermore, according to the OWL's semantics there are two types of Properties: ObjectProperty and DatatypeProperty. ObjectProperty, which can have only Individuals in its range and domain, is represented in OUP as the Class' stereotype «ObjectProperty». DatatypeProperty is modeled with the Class' stereotype «DatatypeProperty». An example of a class diagram that depicts ontology properties modeled in UML is shown in Figure 5. The example means the Wine class has the «ObjectProperty» locatedIn, i.e. the Wine class is the domain of the property locatedIn. Also, the range of «ObjectProperty» locatedIn is the Region class. Since, ontology languages (i.e. OWL and ODM) may additionally specify object properties, we introduced tagged values describing those additional characteristics: *symmetric*, *transitive*, *functional*, and *inverseFunctional*.

In OUP we use the «Restriction» stereotype to refine property's restrictions. As a result, we have an association (e.g. stereotype «someValuesFrom») between a class and an unnamed «Restriction», and two stereotyped dependencies from «Restriction» – «onProperty», and for example, «someValuesFrom» (but stereotypes «hasValue» and «someValuesFrom» can also be used). However, adding this «Restriction» construct in OUP is not the same as adding a class into property domain. Actually, it is mapped as a super class for the given class. Figure 5 depicts a class' restriction on a property – the Wine's «ObjectProperty» locatedIn has as *someValuesFrom* restriction the «OntClass» Region. That means, each instance of the Wine class must have at list one instance of the property locatedIn whose range is the Region class. An additional restriction is multiplicity (i.e. how many property instances can be attached to a class) defined through the multiplicity at the association between the Wine class and the locatedIn property.

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

Since in UML a Class' instance is an Object, in OUP Statement is modeled with Object's stereotype «ObjectProperty» or «DatatypeProperty». 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 cannot have a stereotype. These Links are actually instances of the properties «domain» and «range». In brief, in OUP Statement is represented as an Object with two Links – the subject Link and the object Link, which is shown in Figure 6. Here we have a statement that says the Region's instance MendocinoRegion is locatedIn SonomaRegion, and its

`adjacentRegion` is `CaliforniaRegion`. Unlike other MDA-based approaches to ontology development both ODM and OUP support modeling of body of knowledge (i.e. class instances) [13].

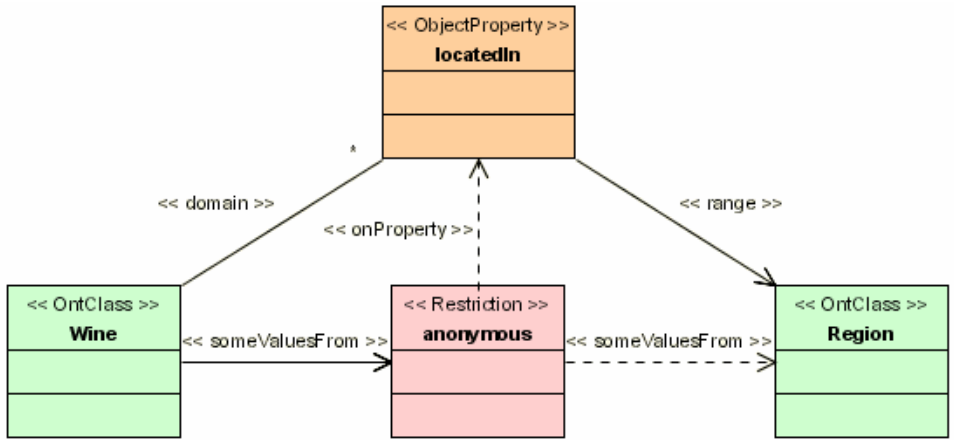

Figure 5 OUP class property and its restriction in the Wine ontology

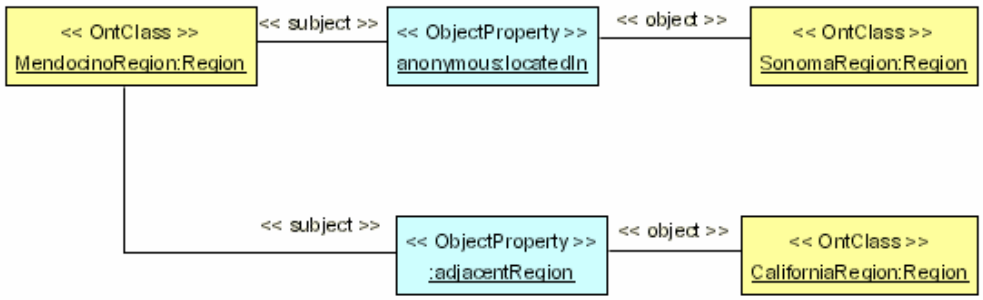

Figure 6 OUP fully supports ontology body of knowledge (i.e. instances) through OUP statements: the Wine ontology example

4 Conceptual solution

The presented ontology languages (ODM and OUP) are MOF-compliant languages defined in the context of the MDA's metamodeling architecture. However, they are not sufficient; they need interaction with real-word ontologies, e.g. with OWL ontologies. It is obvious that we should develop transformations to support conversions between MDA ontology languages and OWL. All these transformations can be explained in the context of technological spaces. So, we firstly identify all technological spaces related to this problem and depict their mutual relations.

4.1 Relations between technological spaces

Figure 7 shows all technological spaces we recognized as important for MDA standards and ontological engineering to be used cooperatively. In the MDA technological space we defined ODM and OUP. It is important to note that ODM is defined at M2 layer, while OUP resides at both M1 and M2 layers according to [1]. Concrete real world models are at M1 layer and they consist of classes and

their instances. That means, we must not have one ontological layer at M1 layer according to [2], and it the case of UML we have two ontological layers: one for classes and one for class instances (i.e. objects). For all MDA layers one can use XMI, an XML-compatible format for sharing metadata.

The OWL technological space includes the W3C's recommendation for the Web Ontology Language. This ontology language is based on XML and RDF(S), and thus an XML format is being used for interchanging OWL ontologies. In this technological space we identify different abstraction layers in order to find relations with the MDA technological space. Two bottom-most layers are denoted O1 and O0. At O1 layer we build ontologies, i.e. we create classes, properties, relations, and restrictions. On the other hand, ontological instances are at O0 layer in the OWL technological space. We use an analogy of the top-most layer defined in the OWL technological space with the results given in [16]. In that paper the authors described an ontology language (i.e. Ontology Inference Layer – OIL) using another ontology language (i.e. RDF(S)). In fact, they created a meta-ontology. Finally, we can say there is a meta-ontology that defines OWL and this meta-ontology is at O2 layer. Note that this three-layer organization of ontologies is already known in AI as Brachman's distinction of knowledge representation systems [11].

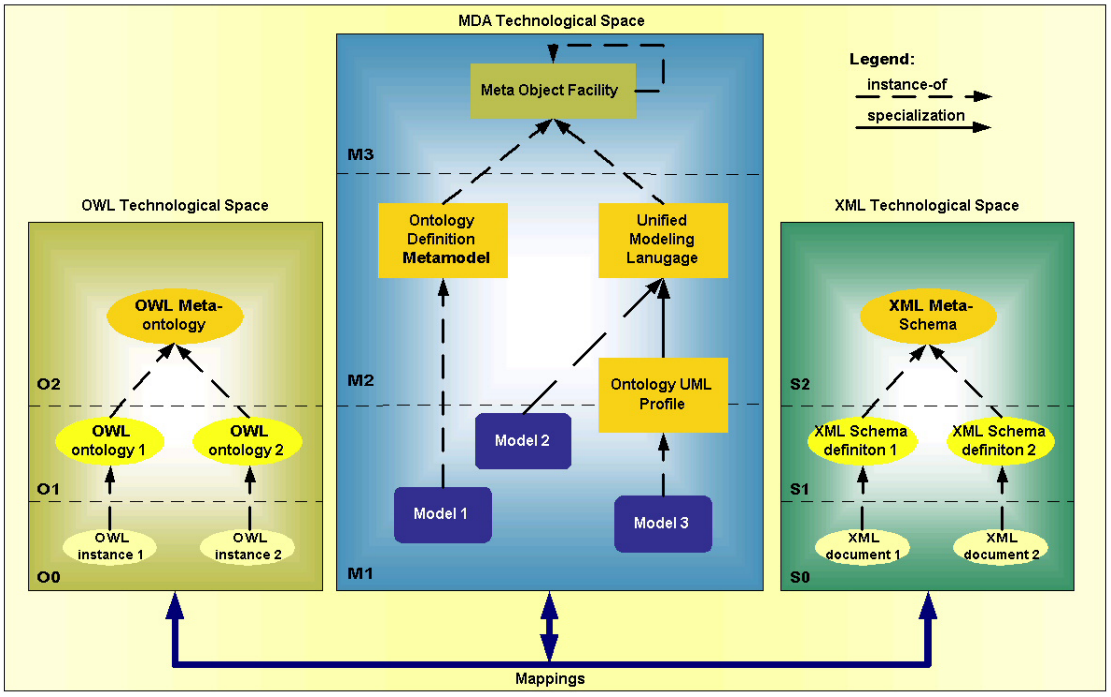

Figure 7 An overview of technological spaces, which are important for the collaborative use of the MDA-compliant ontology languages and OWL, and their mutual relations: MDA, OWL, and XML technological spaces

We can give some important statements in order to provide transformations between these two technological spaces:

1. O2 layer's role (meta-ontology) is similar role to that of the metamodel (M2) layer (e.g. ODM and OUP) – they both specify an ontology language
2. Both O1 and O0 layers have role similar to that of the MDA model (M1) layer. In fact, this conclusion comes form the Atkinson and Kühne's ontological and linguistic layers [2] where one linguistic layer (in this case M1) can contain many ontological layers (in this case O1 and O0). Accordingly, O1 layer is equivalent to M1 classes and O0 layer is equivalent to M1 objects.

Since these two technological spaces both use XML for sharing their metadata, we can include a new technological space in this discussion. Of course, this is the XML technological space, which also has its own layered organization. This organization is very similar to that of OWL, but it is defined in terms of syntax (not semantics) [33]. We observe the XML technological space in terms of W3C XML Schema recommendation. S2 layer is a schema for schemas (i.e. meta-schema) – this schema defines the validity of XML Schema definition documents. Domain specific XML vocabularies (i.e. schemas) are defined at S1 layer. Concrete XML documents are at S0 layer. Epistemologically, one can say that these three layers are equivalent to the OWL layers (i.e. S2 $\Leftrightarrow$ O2, S1 $\Leftrightarrow$ O1, S0 $\Leftrightarrow$ O0). Accordingly, there exists the same relation between MDA layers and XML layers (i.e. M2 with S2, M1 with both S1 and S0). Finally, we can define OWL and MDA languages in terms of the XML Schema. That means we can define schemas that specify the OWL's XML syntax and an XML syntax of a metamodel (e.g. ODM and OUP).

4.2 Transformations between technological spaces

It is obvious from the previous descriptions that we cannot provide direct mappings between the MDA technological space and the OWL technological space. In fact, this transformation can only be defined through the XML technological space. It is important to define a *pair* of transformations in order to enable two-way mapping (one transformation for either direction) between all OWL ontologies and all ontologies represented in an MDA-based ontology language. The transformations can be based on "meta-definitions" of OWL (i.e. on its meta-ontology) and an MDA-compliant language (i.e. a metamodel). This transformation principle is compliant with the Bézivin's principle of metamodel-based model transformation [8]. Practically, in the case of the XML technological space the transformations are based on the XML schemas of both OWL and XMI (i.e. the XML Schema of the UML XMI format). Furthermore, since an XSLT document is a valid XML document we can say that XSLT itself is defined as an XML Schema vocabulary. Figure 8 shows the OUP/OWL transformation of an OUP model (i.e. an OUP document in the XMI format) to its equivalent OWL ontology (i.e. an OWL document in XML format). The transformation maps the MDA M1 layer into its corresponding OWL layers (O1 and O0). The most suitable implementation for this transformation is XSLT, since in this case an XML document is converted into another XML document. The opposite transformation (from OWL to OUP) can also be implemented in XSLT. However, we do not recommend this kind of implementation since we may use different XML representations (e.g. XML Schemas) with different XML tag names to represent the semantics of an OWL ontology and its instances (for details see [16]). For example, the *book* concept can be represented by different tags (*book*, *bookInfo*, etc.), but the ontology defines that all these tags represent the same ontology concept. In this case we suggest using a programming language to implement the transformation. For example, this transformation can be implemented in Java, but Java should be empowered with an OWL parser (e.g. http://jena.sourceforge.net/).

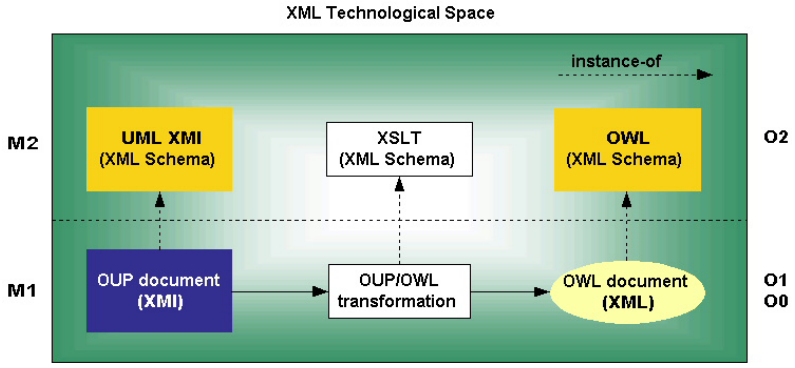

Figure 8 An example of transformation in the XML technological space: the transformation of OUP into OWL

Figure 9 shows an example of transformations in the MDA technological space. This figure is organized according to the transformation schema proposed in [36] for transforming XML schemas to application models. In the MDA technological space we can only transform those ontology languages that have an MDA-compliant metamodel. We illustrate the transformation between OUP and ODM. In terms of the MDA technological space the transformations between these languages should be implemented in one of QVT languages (e.g. ATL [47]), and the chosen QVT language should have its own metamodel. This example shows that there are no problems related to a different metamodeling layer, since both metamodels are defined at the same layer (M2 layer). Actually, the OUP metamodel can reside at both M1 and M2 layers. For the sake of symmetry, we placed the OUP metamodel only at M2 layer in Figure 9. As a matter of fact, this can be true if we would use a standard UML Profile for ontology development without the user's extensions (see [1] for details). Note that this transformation can also be implemented through the XML technological space in terms of XML schemas and XML documents.

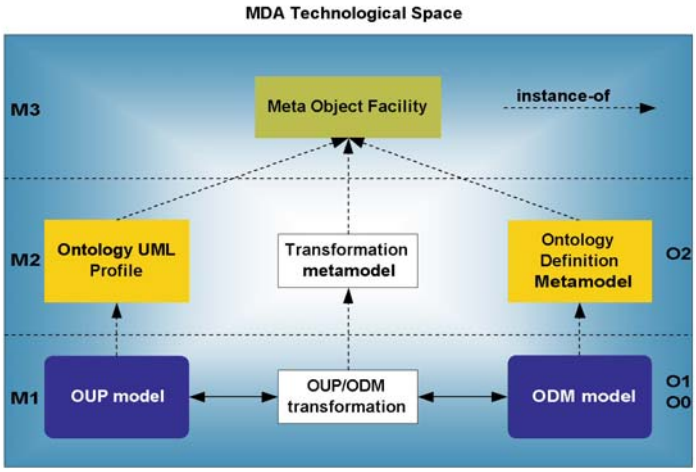

Figure 9 Transformation in the MDA technological space: transformations between OUP and ODM

Summarizing all these facts about transformations between OWL and MDA, Table 2 gives some guidelines on how to make transformation between each pair of the languages discussed above. In the table we only indicate the transformations between each of the observed languages that can be done by

employing one transformation technique. At the first look at the table one might think that transformation between MDA-based languages (i.e. ODM and OUP) and OWL can be done within the MDA technical space. But, it is not possible since OWL is not a part of MDA technical space, and it is not a MOF-based language. That means, the MDA transformation technology (e.g. QVT) cannot be applied to that transformation pair. QVT can only be applied to MOF-based languages. So, we have to look for intersection between those two languages. Since there exist bridges between OWL and XML, and MDA and XML, but not between OWL and MDA, the transformation can only be defined in XML TS as it is shown in Table 2. However, if understand OMG's ongoing model-to-text initiative [44] as a part of MDA technical space, then transformations between ODM and OWL are possible in terms of MDA TS. Since, model-to-text is its initial stage of the development (i.e. there is just a request for proposals at OMG [44]) and there is no any test implementation of that proposal at the moment, we do not consider that case in the table. Consequently, the table should not be understood as a definite list of possible transformations.

Table 2 Overview of possible transformations between OWL, ODM and OUP: technological spaces in which the transformations can be done (XML TS and MDA TS) and implementation technologies for these transformations (XSLT, Query/View/Transformation – QVT, and programmed)

Source language \ Target language	ODM	OUP	OWL
ODM	–	XML TS / XSLT; MDA TS / QVT	XML TS / XSLT
OUP	XML TS / XSLT; MDA TS / QVT	–	XML TS / XSLT
OWL	XML TS / Programmed*, XSLT	XML TS / Programmed*, XSLT	–

* preferred case (e.g. Java empowered with a library for parsing OWL)

5 An implementation example: an XSLT-based approach

In the previous section we explained the conceptual solutions for transforming MDA-based ontology languages (i.e. ODM and OUP) and OWL. In this section we show an implementation example that transforms an OUP-based ontology to its equivalent OWL ontology [27] [28]. That way we develop Semantic Web ontologies that can be used in real-world Semantic Web applications [34].

5.1 Implementation details

The main idea of having a UML Profile for ontology development is to use existing UML tools. In fact, current UML tools (e.g. Rational Rose and Poseidon for UML) mainly support XMI standard [50] – an MDA XML-based standard for metametamodel, metamodel, and model sharing. Since XMI is XML-based, one can employ XSLT to transform XMI documents into target documents that are not XML documents. These target documents can be written in some ontology language, for example OWL. On the other hand, when we use an approach based on XSLT (XSLT principle) we do not need to modify a UML tool; instead, we just apply an XSLT to the output of the UML tool. Accordingly, we can use the well-defined XML/XSLT procedure shown in Figure 10.

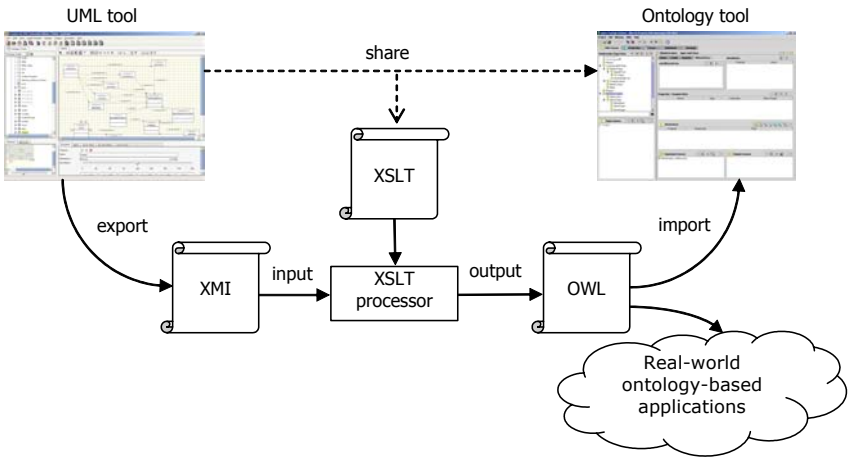

Figure 10 Using XSLT principle: extensions of present UML tools for ontology development

A UML tool can export an XMI document that an XSLT processor (e.g. Xalan – http://xml.apache.org) can use as input. An OWL document is produced as the output, and this format can be imported into a tool specialized for ontology development (e.g. Protégé, http://protege.stanford.edu/), where it can be further refined. On the other hand, since we obtain an OWL document, we do not need to use any ontology tool; the document obtained can be used as the final OWL ontology.

The XSLT we have implemented for mapping OUP models to the OWL ontologies contains a set of rules (i.e. XSLT templates) that match XMI constructs and transform them into equivalent OWL primitives. While developing these rules we had to face some serious obstacles resulting from evident differences between source and target formats. We note some of them:

- The structure of an XMI document is fairly awkward since it contains full description of a UML model. For example, classes, attributes, relations (associations, dependencies, generalization), stereotype descriptions, etc.
- In some cases, OUP uses more than one UML construct to model one OWL element. For example, to model *someValesFrom* restriction using OUP (see Figure 5), we need three UML classes and three relations (i.e. one association and two dependencies). This is especially difficult because each UML construct belongs to a different stereotype.
- UML tools can only draw UML models, but they do not have an ability to check the completeness of an OUP ontology. Thus, the XSLT is incurred to check XMI documents. This is the only way to avoid generation of erroneous OWL ontologies.
- The XSLT must make difference between classes that are defined in other classes (and can not be referenced from other classes using their ID), and classes that can be referenced using their ID. Accordingly, we included into OUP *odm.anonymous* tagged values that help us detect these two cases.

Taking all this into account, it becomes obvious that the developed XSLT is too large to be included in this paper; however, it can be seen at http://www.sfu.ca/~dgasevic/projects/UMLtoOWL/

Figure 11 depicts an output OWL document resulting from applying the XSLT. Figure 11a shows the OWL description of the classes we have defined in Figure 4. Note how OUP classes that have the

tagged value *odm.anonymous* are mapped into OWL (e.g., `WineDescriptor` has an equivalent anonymous class that is defined as a union of `WineTaste` and `WineColor` classes). Figure 11b we shows the OWL description of the `locatedIn` property, which has the `Region` class as its range, and both `Region` and `Wine` classes as its domain. On the other hand, the `Wine` class additionally restricts this property using the OWL *someValuesFrom* restriction. Since OUP has a full support for OWL statements, we are able to transform them into equivalent OWL constructs (i.e. full individual descriptions). Figure 11c contains OWL instances defined as parts of statements in Figure 6. This feature empowers our solution to generate both the ontology armature [17] (classes, properties, etc) and ontology instances (body of knowledge) [13]. This feature is not supported in other MDA-based proposals for ontology development.

The second important decision is how to generate the OWL description, since the same OWL definition (e.g. OWL class) can be generated in more than one way (e.g. an OWL class can be defined using an unnamed class as either equivalent class or subclass). We decided to generate OWL ontologies using a technique similar to the Protégé OWL plugin (http://protege.stanford.edu/plugins/owl/). Hence we have managed to provide an additional way to import Poseidon's models into Protégé through OWL. Of course, since Protégé has more advanced features for ontology development, an OUP-defined ontology can be further improved and refined.

```
<owl:Class rdf:ID="WineDescriptor">
    <owl:equivalentClass>
        <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#WineTaste"/>
                <owl:Class rdf:about="#WineColor"/>
            </owl:unionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:ID="WineTaste">
    <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
</owl:Class>

<owl:Class rdf:ID="WineColor">
    <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <WineColor rdf:about="#Red"/>
                <WineColor rdf:about="#Rose"/>
                <WineColor rdf:about="#White"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

a)

```
<owl:ObjectProperty rdf:ID="locatedIn">
    <rdfs:range rdf:resource="#Region"/>
    <rdfs:domain rdf:resource="#Wine"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Wine">
    <!-- ... -->
    <rdfs:subClassOf rdf:resource="#PotableLiquid"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#locatedIn"/>
            <owl:someValuesFrom
            rdf:resource="#Region"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

b)

```
<Region rdf:ID="SonomaRegion"/>
<Region rdf:ID="CaliforniaRegion"/>
<Region rdf:ID="MendocinoRegion">
    <locatedIn rdf:resource="#SonomaRegion"/>
    <adjacentRegion rdf:resource="#CaliforniaRegion"/>
</Region>
```

c)

Figure 11 Resulting OWL description: a) classes generated for the OUP model from Figure 4; b) Object property OWL descriptors for the model from Figure 5 c) OWL statements from Figure 6

Of course, we should note that Figure 11 is only a part of the OWL description of the Wine ontology obtained by the XSLT. In the next section we outline our first practical experience with this solution.

5.2 Practical experience

We have already noted that the developed transformation acts as an extension to standard UML tools and thus let us create complete OWL ontologies without the need to use ontology-specific development tools. In order to use OUP and the developed XSLT in practice, we should employ an adequate UML tool that supports:

- attaching stereotypes to all UML concepts that we have in OUP. For instance, present UML tools rarely allow objects and link ends to have a stereotype (e.g. objects in Figure 4);
- a convenient way to use tagged values and attach them to each UML element (e.g odm.anonymous tagged value in Figure 4);
- making relations between UML concepts, like those shown in Figure 4. We especially emphasize the importance of relations (e.g. dependency) between a UML class and an UML object. This kind of relation is regular in UML syntax, and can be represented on class diagrams (that are also called static structure diagrams in the UML specification [48]);
- the XMI standard for UML serialization since our XSLT is based on the UML XMI format that transforms to OWL representation in order to create ready-to-use ontologies.

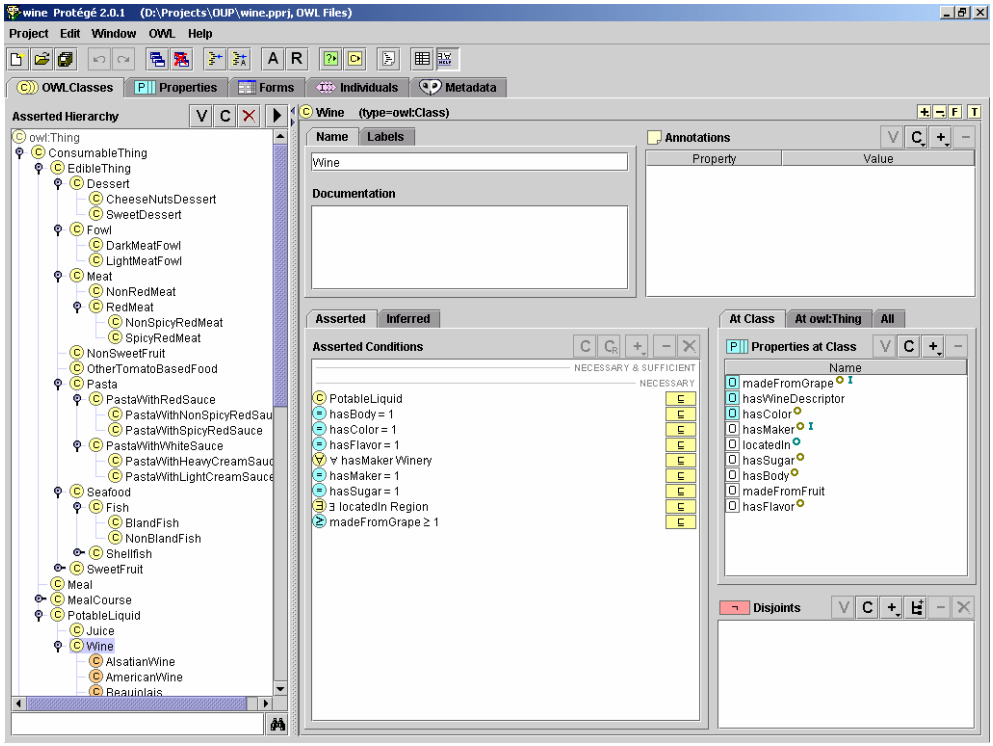

Figure 12 An example of OWL ontology generated from OUP and imported into Protégé: Wine ontology

We have analyzed two UML tools: *IBM/Rational Rose* (a leading UML tool – http://www.rational.com), and *Poseidon for UML* (http://www.gentleware.com). We have decided to use *Poseidon for UML* since it supports all requirements we have mentioned above, unlike *IBM/Rational Rose* that does not provide support for most of them (e.g. object can not have a stereotype, or a class and an object can not be related using any UML's relation). Additionally,

Poseidon for UML is suitable since it uses NetBeans' MDR repository for MOF-compliant metamodel storing (http://mdr.netbeans.org), as well as MOF definition itself. This is an important feature because using model repositories enables us to benefit from all advantages of MDA [10]. From another point of view, this tool also has a closer applicability in ontological engineering since it is a UML tool recommended to be used with Protégé UML backend (http://protege.stanford.edu/plugins/uml) for importing UML models.

We tested our solution using the well-known example of the Wine ontology [40]. Firstly, we represented this ontology in Poseidon using OUP. Parts of this ontology are used in the previous section in order to illustrate OUP (e.g. Figures 3 and 4). Then we exported this extended UML into XMI. After performing the XSLT, we obtained an OWL document. Finally we imported this document into Protégé using its OWL plugin. A screenshot that depicts a part of this imported OWL ontology is shown in Figure 12.

We must admit that we have found a certain difference between OWL generated by the XSLT and OWL produced by Protégé. The difference was detected in the representation of OWL individuals. To represent individuals Protégé uses `owl:Thing` with the attribute `rdf:type` that refers to its type (i.e. its OWL class). For example, `Red` is an instance of the `WineColor` class, and it is represented as follows:

```
<owl:Thing rdf:ID="Red" rdf:type="#WineColor"/>
```

In our solution, an individual is represented using a tag that has the same name as its OWL class. For example, the same Red instance is represented as follows:

```
<WineColor rdf:ID="Red"/>
```

We found this difference unimportant since Protégé is able to recognize OWL instances defined in both forms. Also, these two individual representations have the same meaning in the OWL notation. The current XSLT version has a limitation in that it does not support packages (i.e. multiple OUP ontologies in one UML model). This means, it is unable to produce more than one OWL document (i.e. ontology). Actually, OUP supports multiple ontologies within the same XMI project, but the XSLT standard and XSLT processors introduce this limitation. Of course, this can be overcome using some XSLT's non-standard primitives (i.e. XSLT extensions) that provide multiple documents production from one source XML document (e.g. SAXON XSLT processor and its XSLT extensions).

5.3 Applications

So far, we have developed two ontologies using the OUP that we later transformed in OWL using the XSLT. The two ontologies are the ontology of saints and philosophers and the Petri net ontology. The first has a theological and philosophical character, but also contains a multimedia part that semantically annotates picture collection (icons) [15]. This ontology was developed using the Porphyry's tree method [54] – starting from the Porphyry's tree schema, performing forward classification, and establishing the class hierarchy.

The Petri net ontology was developed in order to provide the Semantic Web support for Petri nets [25]. Petri nets described that way can be inserted into other, non-Petri net XML-based formats, such as *Scalable Vector Graphics* (SVG, the XML-based W3C standard for 2D vector graphics), which makes possible to reconstruct Petri net models using metadata and annotations according to the Petri net ontology. We defined the Petri net ontology using experience from previous Petri net formal descriptions (metamodels, ontologies, and syntax).

6 Related work

In this section we describe existing efforts to enable using UML, current UML tools, as well as MDA-based standards in ontological engineering. Our goal is to explain the formal background of each approach and their mappings to ontology languages. Table 3 summarizes the analyzed frameworks, their formal definitions, the kinds of model interchange description they use, their proposals for implementing the mappings, and the target ontology languages.

The idea to use UML in ontological engineering was firstly suggested by Cranefield [14]. He has found connections between the standard UML and ontology concepts: classes, relations, properties, inheritance, etc. However, there are some dissimilarities between them, and the most important one is related to the property concept – in UML, an attribute's scope is the class that defines it, whereas in ontology a property is a first-class concept that can exist independently of a class. This approach suggests using UML class diagrams for development of ontology taxonomy and relations between ontological concepts, whereas UML object diagrams were intended to be used for modeling ontology instances (i.e. body of knowledge) [13]. Also a practical software support was provided in the form of two XSLTs that were developed to enable transformation of the UML XMI format to RDFS and Java classes. However, we have noticed some limitations as well (that are also propagated to generated languages):

- one cannot conclude whether the same property was attached to more than one class;
- one cannot create a hierarchy of properties;
- target RDFS ontology description does not have advanced restriction concepts (e.g. multiplicity).

Backlawski and his colleagues have introduced two approaches to ontology development. The first one extends the UML metamodel by introducing new metaclasses [3]. For instance, these metaclasses define a property as a first class concept, as well as a restriction on a property. This way they solved the "property problem" in UML. This solution is mainly based on the DAML+OIL ontology language [38]. In order to enable using standard UML tools, they propose a UML profile and its mapping to DAML+OIL. The authors realized that this solution was fairly awkward because it introduced some new concepts in the UML metamodel. Therefore, they have developed an independent ontology metamodel using the MOF, which they named the Unified Ontology Language (UOL) [4]. This metamodel was also inspired by DAML+OIL. We have been unable to find any practical software tool that would be able to map these two MDA-based ontology languages into a Semantic Web language.

Falkovych and her associates [23] do not extend the standard UML metamodel in order to enable transformation of UML models into equivalent DAML+OIL descriptions. They use a UML-separated hierarchy to define the kinds of ontology properties. A practical mapping from UML models to DAML+OIL is implemented using XSLT. The main limitations of this solution are:

1) the lack of mechanisms for formal property specification (e.g. defining property inheritance, or *inverseOf* relation between properties);
2) it is based on UML class diagrams, which contain only graphical artifacts of real UML elements included in a model (e.g. all associations titled with the same name are assumed to represent the same property, although each association is a distinct model element in UML). Of course, this diagram problem can be partly overcome with XMI for UML 2.0 that supports diagram representation.

Protégé is the leading ontological engineering tool [42]. It has a complex software architecture, easily extensible through plug-ins. Many components that provide interfaces to other knowledge-based tools (Jess, Argenon, OIL, PAL constraint, etc.) have been implemented in this way, as well as support for different ontology languages and formats like XML, DAML+OIL (backends), and OIL (tab). In fact, Protégé has a formally defined MOF-based metamodel. This metamodel is extensible and adaptable. This means that Protégé can be adapted to support a new ontology language by adding new metaclasses and metaslots into a Protégé ontology. Introduction of these new metamodeling concepts enable users to add necessary ontology primitives (e.g. the Protégé class has different features from OWL class). In that way it can, for instance, support RDFS [41] or OWL. It is especially interesting that Protégé has backends for UML and XMI. These two backends use the NetBeans' MetaData Repository (MDR – http://mdr.netbeans.org). The first backend exchanges UML models (i.e. classes, and their relations) using the standard UML XMI format, while the second one uses the XMI format that is compliant with the Protégé MOF-defined metamodel. It is obvious that one can share ontologies through Protégé (e.g. import an ontology in the UML XMI format and store it in the OWL format). However, Protégé has one limitation in its UML XMI support – it does not map class relations (i.e. associations) into a Protégé ontology (i.e. it does not attach instance slots to classes). This limitation was expected since Protégé imports UML models without any extension (i.e. a UML Profile).

Table 3 An overview of present UML and MDA based ontology development frameworks and their transformations to the Semantic Web languages

Approach	Metamodel	Model description	Transformation mechanism	Generated ontology language
Cranefield [14]	Standard UML	UML XMI	XSLT	RDFS, Java classes
Backlawski et al [3] [4]	UML Profile, MOF-based ontology language	(not given - UML XMI, and MOF XMI can be used)	-	DAML
Falkovych at al [23]	Standard UML	UML XMI	XSLT	DAML + OIL
Protégé	Protégé metamodel	Protégé XMI	Programmed	OWL, RDF(S), DAML+OIL, XML, UML XMI, Protégé XMI, …
	Standard UML	UML XMI		
DUET	UML Profile	Rational Rose, ArgoUML	Programmed	DAML+OIL
Xpetal	Standard UML	Rational Rose mdl files	Programmed	RDFS

The software tool called DUET (http://codip.grci.com/Tools/Tools.html), which enables importing DAML ontologies into Rational Rose and ArgoUML as well as exporting UML models into the DAML ontology language [24], has been developed in order to support ontological engineering. This tool uses a quite simple UML Profile that contains stereotypes for modeling ontologies (based on UML package) and properties (based on UML class). Additionally, DUET uses an XSLT that transforms RDFS ontologies into equivalent DAML ontologies. That way, an RDFS ontology can be imported into UML tools through the DAML language. Of course, this tool has constraints similar to approaches we have already discussed (e.g. Falkovych et al) since it has no ability to define advanced

class and property relations (e.g. *inverseOf*, *equivalentProperty*, *equivalentClass*, etc.). On the other hand, this is the first UML tool extension that enables ontology sharing between ontology language (i.e. DAML) and a UML tool in both directions.

Xpetal (http://www.langdale.com.au/styler/xpetal) is another tool implemented in Java that transforms Rational Rose models from the *mdl* format to RDF and RDFS. This tool has limitations similar to those that we have already mentioned while discussing Cranefield's software (XSLTs), since it uses only standard UML and does not provide a convenient solution for representing properties, their relations, advanced class restrictions, etc. Actually, this tool is even more limited than the Cranefield's one, since it is oriented to Rational Rose, in contrast to the Cranefield's XSLT that is applicable to every UML XMI document and independent of UML tools.

Our opinion is that all these approaches we have explored above are useful, but none of them gives a full solution that contains:

- a formal description of the new MDA-based ontology language;
- a related UML profile and necessary transformations between these two languages, as well as transformations to contemporary Semantic Web languages (i.e. OWL) [46].

We believe that full usage of MDA provides us with considerable benefits when defining metamodeling architecture and enables us to develop new languages (i.e. ontology language). Actually, there is a RFP at OMG that should enclose all these requirements, but it is still in its initial stage (http://ontology.omg.org).

7 Discussion

Note that the model-sharing principle illustrated in this paper is closely related to a well-proven, general knowledge-sharing mechanism that has already been used in other approaches. For example, Generic Frame Protocol (GFP) was proposed and developed by SRI International and Stanford University long before the concepts of XML and XSLT were established, in order to provide a generic model for frame representation and, in fact, generic interface to different frame representation systems (FRS) [32]. Essentially, GFP provided generic knowledge-base functions for representing and manipulating knowledge in FRS, and a translation layer between these functions and existing FRS-specific functional interfaces. The role analogous to that of XSLT in our case was given to a set of translators provided by FRS developers – these translators ensured translation between FRS-specific representation languages and the language of the GFP. GFP has later evolved into the Open Knowledge Base Connectivity (OKBC) standard application-programming interface (implemented in several different languages) for accessing knowledge bases stored in different knowledge representation systems [43].

Another feature can be introduced in this approach, since we have a formal metamodeling specification for ontology development, defined by MOF. That means MDA-based repositories can be used for storing metamodels and models. If we use present MDA-based repositories we can produce *Java Metadata Interface (JMI)* [18] compliant code, and thus obtain a possibility to incorporate a programming logic into Java applications. The JMI Specification defines the Java programming interface for manipulating MOF-based models and metamodels. JMI also enables generation of programming interfaces based on such models. This feature can be used for both ODM- and OUP-compliant models, since they are both defined using MOF (OUP is also defined using MOF because it

is a UML extension). Accordingly, we have implemented a solution for ontology development that uses the MDR – a NetBeans' repository – and can produce JMI.

One very important remark is that the implemented XSLT (see Section 5), in fact, is not part of OMG's RFP for Ontology definition metamodel [46]. This document presumes transformations between ODM and OUP, as well as transformations between ODM and OWL (see Figure 13). This means, if one wants to transform an OUP-defined ontology into OWL, that ontology should firstly be transformed into ODM, and subsequently from ODM to OWL. Of course, it is also possible to implement all using XSLT because all ontology representations use XML: ODM uses XMI format – MOF-defined metamodel, OUP uses the UML XMI format, and OWL has an XML-based representation. Our transformation from OUP to OWL is a practical extension of present UML tools that gives them capability to be used for full development of ontology described by a real Semantic Web language. It is a kind of a bridge between ontological and software engineering, since current MDA-compliant implementations are in very immature stage. Development of these ODM↔OUP and ODM↔OWL transformations is currently our primary activity.

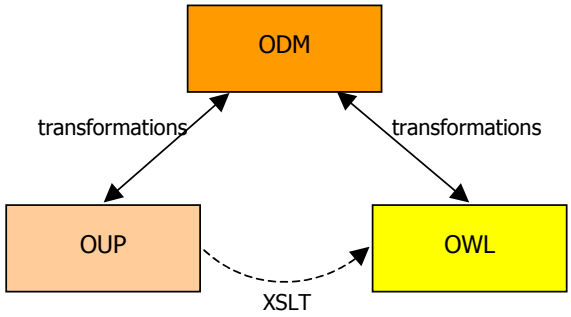

Figure 13 Relations between the implemented solution and recommended transformations in OMG's RFP

Transformations from OUP to ODM, and from ODM to OWL offer the following advantages:

- When one wants to support a new ontology language (e.g. DAML+OIL) using the ODM-based principle, only a pair of transformations should be implemented: from a new language to the ODM, and from the ODM to a new language. In the case we want to support transformations between N different languages (like OUP and OWL), then it is necessary to implement 2N transformations. However, if we implement transformations between each pair of ontology languages without ODM (e.g. OWL and DAML+OIL) then we need N^2 transformations.
- Since we should transform all ontologies through ODM, we can validate an ontology against the ontology metamodel (i.e. ODM). This way, we can prevent transformation of an invalid ontology or issue a warning when an ontology is inconsistent. This feature is very important for OUP models since existing UML tools are unable to check UML models that are based on a UML Profile.
- Finally, we can say that the transformation mechanism for MDA-based ontologies is driven by the ideas from [8] in which the author proposes metamodel-based model transformations. In the case of MDA-based ontology languages we have different metamodels (i.e. OWL, OUP metamodel). However, the ODM serves as an integration point that decreases the number of needed transformations. Also, we can prove usefulness of having a central metamodel (in this case, ODM) for some area.

8 Conclusions

In this paper we showed a formal approach to bringing closer MDA-based ontology languages and the W3C OWL recommendation using the idea of technological spaces. Firstly, we gave a set of definitions regarding the MDA concept: metamodeling, the layered metamodeling architecture, UML Profiles, and the XMI format. Then we identified epistemological relations between MDA-defined ontology metamodels and OWL. Accordingly, we deduced the equivalences between their layers as follows: MDA M2 layer (metamodel) is equivalent to OWL O2 layer (meta-ontology), whereas M1 layer corresponds to both O1 and O0 layers. Finally, we showed that their mutual mappings could be done through the XML technological space. Having followed these results we implemented a transformation, which converts an ontology UML Profile into OWL, using XSLT.

We believe that results given in this paper can be useful to the researchers from the Semantic Web community who are trying to benefit from ontology development with MDA standards. Especially, this is important for the future software tools supporting these efforts. We hope that this work can be useful as a practical contribution to the OMG efforts in finding a suitable MDA-based technique for the Semantic Web ontologies, which will bring ontology development process closer to software engineers.

In the future, we will finish our current work aimed at providing support for transformations between the Ontology UML Profile (i.e. the UML XMI format) and the Ontology Definition Metamodel (i.e. the ODM specific XMI format), as well as between OWL and Ontology Definition Metamodel. In this way, we will have an entire metamodeling platform compliant with the OMG's ontology initiative. On the other hand, we face many research challenges such as: exploring possibilities for developing a graphical tool that will be able to construct transformations for bridging different technological spaces (a tool should be implemented to work in XML and MDA technological spaces); and analyzing usefulness of a new language for transforming actual ontology languages (e.g. OWL) which will have a similar role for ontology languages as XSLT has for XML.

References

[1] Atkinson, C. and Kühne, T., Profiles in a strict metamodeling framework, Science of Computer Programming, 44(1), 2002. 5-22.

[2] Atkinson, C. and Kühne, T., Model-Driven Development: A Metamodeling Foundation. IEEE Software 20(5), 2003. 36-41.

[3] Baclawski, K., Kokar, M.K., Kogut, P., Hart, L., Smith, J.E., Letkowski, J. and Emery, P. Extending the Unified Modeling Language for ontology development. International Journal Software and Systems Modeling (SoSyM), 1(2), 2002. 142-156.

[4] Baclawski, K. Kokar, M., Smith, J.E., Wallace, E., Letkowski, J., Koethe, M.R. and Kogut, P. UOL: Unified Ontology Language. Assorted papers discussed at the DC Ontology SIG meeting, (2002), http://www.omg.org/cgi-bin/doc?ontology/2002-11-02

[5] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L. Patel-Schneider, P.F. and Stein, L.A. OWL Web Ontology Language Reference. W3C Recommendation, (2004), http://www.w3.org/TR/2004/REC-owl-ref-20040210

[6] Berners-Lee, T. Weaving the Web. Orion Business Books, 1999.

[7] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J.E., First experiments with the ATL model transformation language: Transforming XSLT into Xquery. in Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, (Anaheim, CA, USA, 2003).

[8] Bézivin, J., From Object Composition to Model Transformation with the MDA. in Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, (Santa Barbara, USA, 2001), 350-355.

[9] Bézivin, J., In Search of a Basic Principle for Model Driven Engineering. Upgrade, 5(2), 2004. 21-24.

[10] Bock, C. UML without Pictures. IEEE Software, 20(5), 2003. 33-35.

[11] Brachman, R.J. On the Epistemological Status of Semantic Networks, in Findler, N.V. ed. Associative Networks: Representations and Use of Knowledge by Computers, Academic Press, 1979, 3-50.

[12] Brickley, D. and Guha, R.V. (eds.) Resource Description Framework (RDF) Schema Specification 1.0. W3C Recommendation, (2004), http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[13] Chandrasekaran, B., Josephson, J.R. and Benjamins, V.R. What Are Ontologies, and Why Do We Need Them?. IEEE Intelligent Systems, 14(1), 1999. 20-26.

[14] Cranefield, S. Networked Knowledge Representation and Exchange using UML and RDF. Journal of Digital information, 1(8), 2001. http://jodi.ecs.soton.ac.uk

[15] Damjanović, V. Semantic Web, Ontologies, and Agents. Specialist degree thesis, University of Belgrade, Serbia and Montenegro (2003).

[16] Decker, S., Melnik, S., van Harmelen, F., Fensel, D., Klein, M., Broekstra, J., Ederman, M. and Horrocks, I. The Semantic Web: The Roles of XML and RDF. IEEE Internet Computing, 4(5), 2000. 63-74.

[17] Devedžić, V. Understanding Ontological Engineering. Communications of the ACM, 45(4), 2002. 136-144.

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

[19] Djurić, D., MDA-based Ontology Infrastructure. International Journal on Computer Science and Information Systems, 1(1), 2004. 91-116.

[20] Djurić, D., Gašević, D., Devedžić, V. and Damjanović, V., A UML profile for OWL ontologies. in Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, (Linköping, Sweden, 2004).

[21] Djurić, D., Gašević, D. and Devedžić, V. Ontology Modeling and MDA. Journal on Object Technology, 4(1), 2005. 109-128.

[22] Duddy, K. UML2 Must Enable A Family of Languages. Communications of the ACM, 45(11), 2002. 73-75.

[23] Falkovych, K., Sabou, M. and Stuckenschmidt, H. UML for the Semantic Web: Transformation-Based Approaches. in Omelayenko, B. and Klein, M. eds. Knowledge Transformation for the

Semantic Web, Frontiers in Artificial Intelligence and Applications, Vol. 95, IOS Press, 2003, 92-106.

[24] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L. and Patel-Schneider, P.F. OIL: An Ontology Infrastructure for the Semantic Web. IEEE Intelligent Systems, 16(2), 2001. 38-45.

[25] Gašević D. and Devedžić, V., Reusing Petri Nets Through the Semantic Web. in Proceedings of the 1st European Semantic Web Symposium, (Heraklion, Greece, 2004) 284-298.

[26] Gašević, D., Damjanović, V. and Devedžić, V., Analysis of the MDA Standards in Ontological Engineering. in Proceedings of the 6th International Conference of Information Technology, (Bhubaneswar, India, 2003), 193-196.

[27] Gašević, D., Djurić, D., Devedžić, V. and Damjanović, V., Converting UML to OWL ontologies. in Proceedings of the 13th International WWW Conference, (NY, USA, 2004) 488-489.

[28] Gašević, D., Djurić, D., Devedžić, V. and Damjanović, V., UML for Read-To-Use OWL Ontologies. in Proceedings of the 2nd IEEE International Conference on Intelligent Systems, (Vrana, Bulgaria, 2004) 485-490.

[29] Gómez-Pérez, A. and Corcho, O. Ontology Languages for the Semantic Web. IEEE Intelligent Systems, 17(1), 2002. 54-60.

[30] Gruber, T. A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2), 1993. 199-220.

[31] Juerjens, J. Secure Systems Development with UML. Springer-Verlag, 2003.

[32] Karp, P.D., Myers, K. and Gruber, T., The Generic Frame Protocol. in Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI 1995, (Montréal, Québec, Canada, 1995), 768-774.

[33] Klein, M. XML, RDF, and Relatives. IEEE Intelligent Systems, 16(2), 2001. 26-28.

[34] Klein M. and Visser, U. Guest Editors' Introduction: Semantic Web Challenge 2003. IEEE Intelligent Systems, 19(3), 2004. 31-33.

[35] Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M. and Smith, J. UML for Ontology Development. The Knowledge Engineering Review, 17(1), 2002. 61-64.

[36] Kurtev, I. and van den Berg, K., Model Driven Architecture based XML Processing. in Proceedings of the ACM Symposium on Document Engineering, (Grenoble, France, 2003), 246-248.

[37] Kurtev, I., Bézivin, J. and Aksit, M., Technological Spaces: An Initial Appraisal. in Proceedings of the Confederated International Conferences CoopIS, DOA, and ODBASE 2002, Industrial track, (Irvine, CA, USA, 2002).

[38] McGuinness, D.L., Fikes, R., Hendler, J. and Stein, L.A. DAML+OIL: An Ontology Language for the Semantic Web. IEEE Intelligent Systems, 17(5), 2002. 72-80.

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

[40] Noy, N.F. and McGuinness, D.L., Ontology Development 101: A Guide to Creating Your First Ontology. Knowledge Systems Laboratory, Stanford University, (2004) http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html

[41] Noy, N.F., Fergerson, R.W. and Musen, M.A., The knowledge model of Protégé-2000: combining interoperability and flexibility. in Proceedings of the 12th International Conference, (Juan-les-Pins, France, 2000), 17-32.

[42] Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W. and Musen, M.A. Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems, 16(2), 2001. 60-71.

[43] Open Knowledge Base Connectivity, (2003) http://www.ai.sri.com/~okbc/

[44] MOF Model to Text Transformation Language - Request For Proposal. OMG Document ad/04-04-07, (2004), http://www.omg.org/cgibin/apps/doc?ad/04-04-07.pdf

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

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

[47] MOF 2.0 Query/Views/Transformations Request for Proposal. OMG Document ad/2002-04-10, (2002), http://www.omg.org/docs/ad/02-04-10.pdf

[48] Unified Modeling Language Specification v1.5. OMG Document formal/03-03-01, (2003), http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.zip

[49] Unified Modeling Language: Superstructure, Version 2.0, Final Adopted Specification, OMG Document ptc/03-08-02, (2003), http://www.omg.org/cgi-bin/apps/doc?ptc/03-08-02.zip

[50] OMG XMI Specification, v1.2. OMG Document formal/02-01-01, (2002), http://www.omg.org/cgi-bin/doc?formal/2002-01-01

[51] Seidewitz, E. What Models Mean. IEEE Software, 20(5), 2003. 26-32.

[52] Selic, B. The Pragmatics of Model-Driven Development. IEEE Software, 20 (5), 2003. 19-25.

[53] Sigel, J. Developing in OMG's Model-Driven Architecture, Revision 2.6. OMG's White Paper, (2001), ftp://ftp.omg.org/pub/docs/-omg/01-12-01.pdf

[54] Sowa, J.F. Knowledge Representation: Logical, Philosophical, and Computational Foundation. Brooks Cole Publishing Co., 2000.