Int J Softw Tools Technol Transfer
DOI 10.1007/s10009-006-0002-1

REGULAR CONTRIBUTIONS

MDA-based automatic OWL ontology development

Dragan Gašević · Dragan Djurić · Vladan Devedžić

© Springer-Verlag 2006

Abstract This paper presents an eXtensible Stylesheet Language Transformation (XSLT)-based approach for automatic generation of the Web Ontology Language (OWL) from a UML model. Similar solutions that treat this problem are mostly partial since they do not use full metamodeling potentials. Although they emphasize the notion of the use UML for ontology development and propose necessary transformations into Semantic Web languages (e.g., RDF Schema, DAML, DAML+OIL), their UML models must be further refined using ontology-specialized tool. None of these approaches enables instance modeling and generation of OWL ontologies. In our efforts to make ontological and software engineering techniques closer, we have firstly defined ontology metamodeling architecture using Model Driven Architecture (MDA) concepts. This architecture consists of the Ontology Definition Metamodel defined using Meta Object Facility (MOF) and based on the OWL, as well as the related Ontology UML Profile (OUP). A transformation, that we present here, extends this metamodeling architecture and transforms an ontology from its OUP definition (i.e., XML Metadata Interchange – XMI) into the OWL description. Accordingly, we illustrate how an OUP-developed ontology can be shared with ontological engineering tools (i.e., Protégé).

Keywords Ontology development · Model Driven Architecture · UML Profile · OWL · XSLT

D. Gašević (✉)
School of Interactive Arts and Technology,
Simon Fraser University Surrey,
2400 Central City, 10153 King George Hwy.,
Surrey, BC V3T 2W1, Canada
e-mail: dgasevic@acm.org

D. Djurić · V. Devedžić
FON – School of Business Administration,
University of Belgrade, POB 52,
Jove Ilića 154, 11000 Belgrade,
Serbia and Montenegro
e-mail: dragandj@gmail.com

V. Devedžić
e-mail: devedzic@fon.bg.ac.yu

1 Introduction

The Semantic Web initiative tries to establish better semantic connections between different resources on the Web using AI techniques. Domain ontology is the most prominent part of this research that should provide a formal way to represent a conceptualization of some domain [22]. 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.) [21]. Even though Semantic Web languages use the 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.

In order to bring ontology development process closer to wider practitioners' population, some authors propose usage of software engineering techniques, especially the UML since it is the most accepted software engineering standard [25]. But, the UML is based on object oriented paradigm, and has some limitation regarding ontology development. However, none of current solutions to this problem supports full ontology

definition: definition of non-limited degree of property hierarchy, modeling ontology instances, mapping multiplicity constraints, definition of additional property constraints (e.g., a property has a value) as well as inverse properties. Hence, we can only use the UML in initial phases of an ontology development. We believe that these limitations can be overcome by using the UML's extensions (i.e., UML profiles) [16], as well as other OMG's standards (e.g., Model Driven Architecture–MDA). Additionally, if we want to provide 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 [41]. Since software industry shows an increasing interest in MDA recently, ontology development may benefit by having support for state-of-the-art industrial software engineering efforts.

Accordingly, we have implemented an eXtensible Stylesheet Language Transformation (XSLT) that transforms the XML Metadata Interchange (XMI) representation of a UML Profile (i.e., The Ontology UML Profile - OUP) into the Web Ontology Language (OWL) [3]. With this UML's model transformation we can extend present UML tools, so that they can be used for full ontology development without need for other ontological tools (e.g., Protégé, OilEd, etc.). This solution is a part of the GOOD OLD AI research group (http://good-oldai.org.yu) efforts to develop an ontology metamodeling architecture *based on the current OMG's initiative* (i.e., Request for Proposal – RFP) for Ontology Definition Metamodel (ODM) [36]. The aim of this initiative is to define a metamodeling architecture for ontology development.

At this stage, we can say that the proposed solution is just one step towards enabling MDA-based ontology development. However, there are still some issues that can only be resolved empirically in the future, such as: will diagram based methods be suitable for developing very large ontologies; and will users familiar with UML know to develop ontologies even though they do not have any previous experience in ontology development.

In Sect. 2 we give an overview of previous work in transforming UML-based models into ontology languages. Section 3 presents a formal framework of our solution – a metamodeling architecture as well as the OUP. In Sect. 4 we show implementation details, whereas Sect. 5 illustrates our experiences with XSLT in sharing ontologies between UML tools (i.e., Poseidon for UML) and ontology development tools (i.e., Protégé), and gives a comparison of their OWL ontology description. Section 6 further discusses the developed solution in order to find better application of the MDA standards in ontology development.

2 Previous work

In this section, we describe existing efforts to enable the use of the UML, present 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 into ontology languages. In Table 1, we give an overview of the analyzed solutions, their formal definition, kinds of model interchange description they use, proposals for mapping implementation, and target ontological languages.

The idea to use the UML in ontological engineering has firstly been in Cranefield's papers [10]. He has found connections between the standard UML and ontologies 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 the UML an attribute has a class scope, while 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) [8]. In its initial development stage, Cranefield's approach used the core UML definition. 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 noted some limitations (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, the target RDFS ontology description does not have advanced restriction concepts (e.g., multiplicity).

Baclawski and colleagues have introduced two approaches for ontology development. The first one extends the UML metamodel by introducing new metaclasses [1]. For instance, these classes define a property as a first class concept, as well as a restriction on a property. In this way, they have solved the "property problem" in the UML. This solution is mainly based on the DAML+OIL ontology language [26]. In order to enable usage of 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

Table 1 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's	Basic UML	UML XMI	XSLT	RDFS, Java classes
Baclawski et al	UML Profile, MOF-based ontology language	(not given - UML XMI, and MOF XMI can be used)	–	DAML
Falkovych at al	Basic UML	UML XMI	XSLT	DAML + OIL
Protégé	Protégé metamodel	Protégé XMI	Programmed	OWL, RDF(S), DAML+OIL, XML, UML XMI, Protégé XMI, ...
	Basic UML	UML XMI		
DUET	UML profile	Rational Rose, ArgoUML	Programmed	DAML+OIL
Xpetal	Basic UML	Rational Rose mdl files	Programmed	RDFS

ontology metamodel by using the Meta-Object Facility (MOF), which they named the Unified Ontology Language (UOL) [2]. This metamodel was also inspired by the DAML+OIL. We have been unable to find any practical software solution that would be able to map these two MDA-based ontology languages into a Semantic Web language.

Falkovych et al. [17] do not extend the standard UML metamodel in order to enable transformation of UML models into equivalent DAML+OIL descriptions. They use an UML-separated hierarchy to define kinds of ontology properties. A practical mapping from UML models to DAML+OIL is implemented by using the XSLT. The main limitations of this solution are (1) 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., they assume all association that has the same name as the same property, even though each association is a distinct model element in the UML). Of course, this diagram problem can be partly overcome with the XMI for the UML 2.0 that supports diagram representation.

Protégé is the leading ontological engineering tool [30]. 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 MOF-defined metamodel. This metamodel is extensible and adaptable. This means, Protégé can be adapted to support a new ontology language by adding new metaclasses and metaslots into a Protégé's ontology. Introduction of these new metamodeling concepts enabled users to add necessary ontology primitives (e.g., the Protégé class has different features from the OWL class). In that way it can, for instance, support RDFS [29] or OWL (http://protege.stanford.edu/plugins/owl- plugin). It is especially interesting that Protégé has backends for the UML and XMI. These two backends use 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 the Protégé (e.g., import an ontology in the UML XMI format and store it in the OWL/XML format). However, Protégé has one limitation in its UML XMI support – it does not map class relations (i.e., associations) into a Protégé's ontology (i.e., does not attach instance slots to classes). But, this limitation was expected since Protégé imports UML models without any extension (i.e., a UML Profile).

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 [18], 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, the DUET uses an XSLT that transforms RDFS ontologies into equivalent DAML ontologies. In 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 discussed so far (e.g., Falkovych et al.) since it has no ability to define advanced class and property relations (e.g., inverseOf, equivalent Property, 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 (i.e., XSLT), since it uses the standard UML and does not provide a convenient solution for representing properties, their relations, advanced class restrictions, etc. Actually, this tool is more limited than Cranefield's one, since it is oriented to Rational Rose, in contrast to 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). We believe that full usage of the recent OMG's effort – Model Driven Architecture (MDA) [27] provides us with considerable benefits when defining metamodeling architecture and enables us to develop new languages (i.e., ontology language). Actually, there was an RFP for Ontology Definition Metamodel (ODM) at OMG that should enclose all these requirements. The four separate ODM proposals responded to OMG's ODM RFP [36] submitted by the following OMG members: IBM [34], Gentleware [33], DSTC [32], and Sandpiper Software Inc and KSL [37]. However, none of those submissions gave a comprehensive proposal. For example, none of them proposed XMI bindings for ODM, none of them proposed mappings between ODM and OWL, only IBM [34] and Gentleware proposed an Ontology UML Profile, etc. Accordingly, the OMG partners decided to join their efforts, and the current result of their efforts is the ODM joint submission [31]. So far, the problem of transformations between MDA-based and ontology languages is not discussed in the submitted document. Furthermore, there is no any practical implementation based upon the OMG's submission. In order to have such a solution we have decided to use our proposal for the MDA-based architecture complying with ODM requirements. We briefly depict this architecture in Sect. 3, with the main focus on the UML profile, which can be used in standard UML tools. On top of that architecture we present the first practical implementation of the transformation between an ODM-based UML Profile and the OWL language.

3 Formal framework of our solution

MDA and its four-layer architecture provide a solid basis for defining metamodels of any modeling language, so it was a straight choice to define an ontology-modeling language in MOF. Such a language can profit from MDA's support in modeling tools, model management and interoperability with other MOF-defined metamodels. Present software tools do not implement many of the MDA basic concepts [20]. However, most of these applications, currently primarily oriented toward the UML and M1 (i.e., model) layer, are expected to be enhanced in the next few years to support MDA.

3.1 Ontology metamodeling architecture

Currently, there is an initiative (i.e., RFP) within the OMG aiming to define a suitable language for modeling Semantic Web ontology languages in the context of MDA [27]. According to this RFP, we present our proposal of such architecture. In this approach to ontology modeling in the scope of MDA, which is shown in Fig. 1, several specifications should be defined:

- 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 Ontology UML Profile and from Ontology UML Profile to other UML profiles.

ODM should be designed to enclose 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 W3C's recommendation. It is at the Logical layer of the Semantic Web [4], on top of RDF Schema (Schema layer). In order to make use of graphical modeling capabilities of the UML, ODM should have a corresponding UML Profile [42]. This profile enables graphical editing of ontologies using UML diagrams as well as other benefits resulting from the use of mature UML CASE tools. Both UML and ODM models are serialized in the XMI format, so that the two-way transformation between them can be done using XSLT. OWL also has representation in the XML format, so another pair of XSLTs should be provided for the two-way mapping between ODM and OWL. For mapping from the Ontology UML Profile into other, technology-specific UML Profiles, additional transformations can be added to support the use of ontologies in the design of other domains and vice versa.

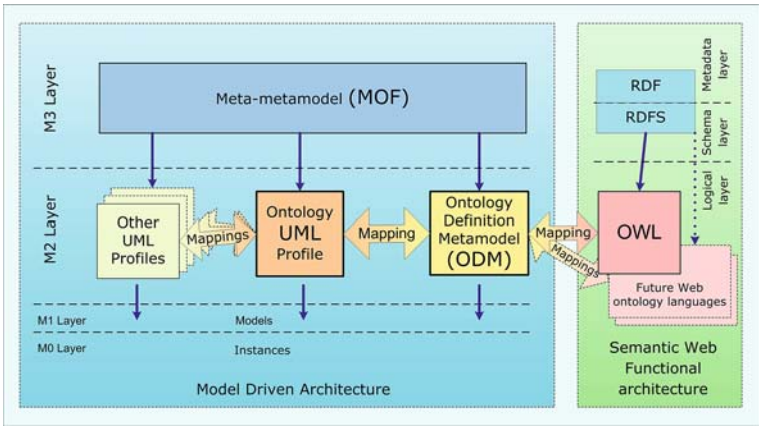

Fig. 1 Ontology modeling in the context of MDA and the Semantic Web

Table 2 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
`Classifier`	Abstract concept that defines classification. It is specialized by `Class`, `DataType`, etc.		In RDF(S), `rdfs:Class` also have function that is similar to a MOF concept of `Classifier`
`Association`	Expresses relationships in the metamodel between pairs of instances of `Class`	`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`

Firstly, we defined ODM using MOF. A brief comparative description of the most important metamodeling constructs in MOF and RDF(S), is shown in Table 2. A detailed description of MOF can be found in the OMG's MOF specification document [35]. RDF, RDFS and their concepts are described in detail in W3C documents [7]. ODM gives us a metamodel-based semantic foundation [40] for ontology languages, so that we can use MDA's capabilities for ontology development. But, if we want to use standard CASE tools for ontology development we need a UML Profile whose formal semantic is in accordance with ODM. Thus, in Sect. 3.2, we briefly outline the ODM-based UML Profile – OUP.

3.2 Ontology UML profile: source language

The basic UML constructs (model elements) can be customized and extended with new semantics using four UML extension mechanisms defined in the UML Specification [38]: stereotypes, tag definitions, tagged values, and constraints. *Stereotypes* enable defining virtual sub-

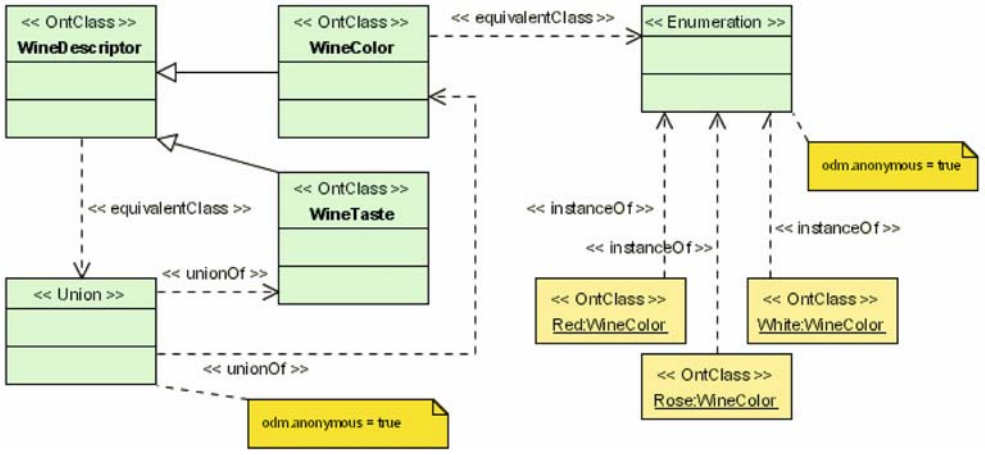

Fig. 2 Class-oriented stereotypes of the Ontology UML profile: An excerpt of the Wine ontology

classes of UML metaclasses, assigning them additional semantics. Here, we briefly outline OUP, whereas its details are given in [15].

In development of our Ontology UML Profile, we used experiences of other UML Profile designers (e.g., see [23]). 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.

`Class` is one of the most fundamental concepts in ODM and OUP. 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`). 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 OUP 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 by using UML stereotypes. In Fig. 2, we show a part of the well-known Wine ontology. `WineDescriptor` is equivalent to the union of classes `WineTaste` and `WineColor`, whereas `WineColor` is an enumeration of `WineColor` instances: `White`, `Rose`, and `Red`. We should note that we have two anonymous classes (`Union` and `Enumeration`). That means that these classes are defined through other classes (e.g., anonymous `Enumeration` is defined in the class `WineColor`) and cannot be used out of their definitions. We use the tag value *odm.anonymous* with a value true, to mark anonymous classes. The users have to manually attach this tagged value to anonymous classes. This helps us differentiate between anonymous and non-anonymous classes in automatic transformation of OUP models.

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 the OUP, `Individual` is modeled as UML `Object`, which is shown in Fig. 2. Here, we had difficulties with decision which stereotype to attach to UML's objects to make them represent ODM's individuals. It would be natural to have a stereotype with the name ≪`Individual`≫, but UML's specification [38] 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 OUP's instances.

Since `Property` is a stand-alone concept it can be modeled using a stand-alone concept in UML. That concept could be UML `Class'` stereotype ≪`Property`≫. However, `Property` must be able to represent relationships between `Resources` (`Classes`, `Datatypes`, etc. in the case of UML), which UML `Class` alone is not able to do. ODM defines two types (subclasses) of `Property`–`ObjectProperty` and `DatatypeProperty`. `ObjectProperty`, which can

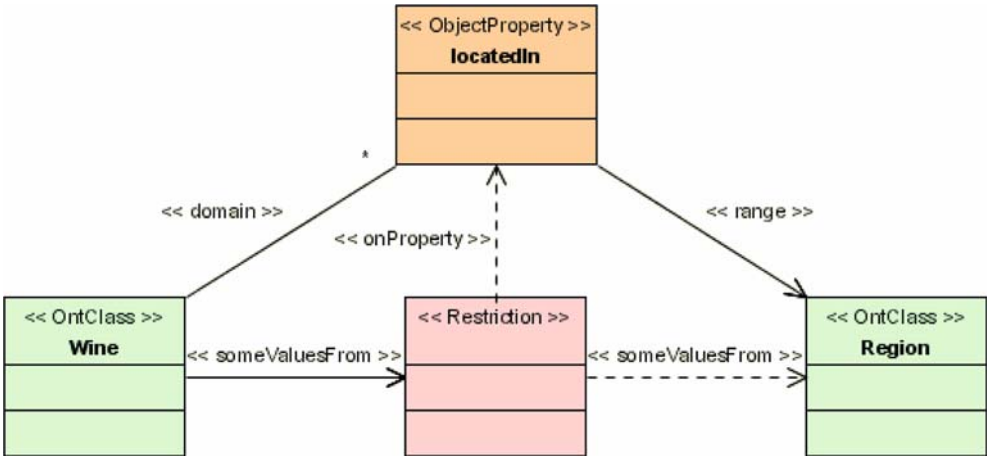

Fig. 3 The OUP class property and restriction on the example of the Wine ontology

have only `Individuals` in its `range` and `domain`, is represented in OUP as `Class'` stereotype ≪`ObjectProperty`≫. `DatatypeProperty` is modeled with `Class'` stereotype ≪`DatatypeProperty`≫. An example of a Class Diagram that depicts ontology properties modeled in the UML is shown in Fig. 3. Tagged values describe additional characteristics of ≪`ObjectProperty`≫: symmetric, transitive, functional,and inverseFunctional.

In OUP we use the ≪`Restriction`≫ stereotype to refine a property's restrictions. As a result we have an association (e.g., stereotype ≪`someValues From`≫) between a class and an unnamed ≪`Restriction`≫, and two stereotyped dependencies from ≪`Restriction`≫– ≪`onProperty`≫, and e.g., ≪`someValues From`≫ (but stereotypes ≪`hasValue`≫ and ≪`allValuesFrom`≫ 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 (i.e., the class Wine). Fig. 3 depicts a class' restriction on a property – the `Wine`'s ≪`ObjectProperty`≫ `locatedIn` has a *someValuesFrom* restriction the ≪`OntClass`≫ `Region`. Association stereotype ≪`domain`≫ defines a property domain, while Association stereotype ≪`range`≫ defines a property range. An additional restriction is its multiplicity (i.e., how many property instances can be attached to a class).

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

Since in the UML `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` can not have a stereotype. These `Links` are actually instances of 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 Fig. 4. Here, we have a statement that says the `Region`'s instance `MendocinoRegion` is `locatedIn SonomaRegion`, and its `adjacentRegion` is `CaliforniaRegion`. Unlike other analyzed MDA-based solutions for ontology development, ODM and OUP support modeling of body of knowledge (i.e., class instances) [8].

4 Overview of our solution: XSLT

In Sect. 3, we have explained MDA-based languages for ontological engineering. The main idea of having a UML profile for ontology development is to use present UML tools. In fact, current UML tools (e.g., Rational Rose, Poseidon for UML) mainly support the XMI standard [39] – MDA's XML-based standard for sharing metametamodels, metamodels, and models. Since this format is XML-defined, one can employ XSLT to transform XMI documents into target documents that are not necessarily XML documents. These target documents can be written in some ontology language, e.g., OWL. On the other hand, when we use an approach based on XSLT (XSLT principle) we do not need to change a UML tool, instead we just apply an XSLT on an output document

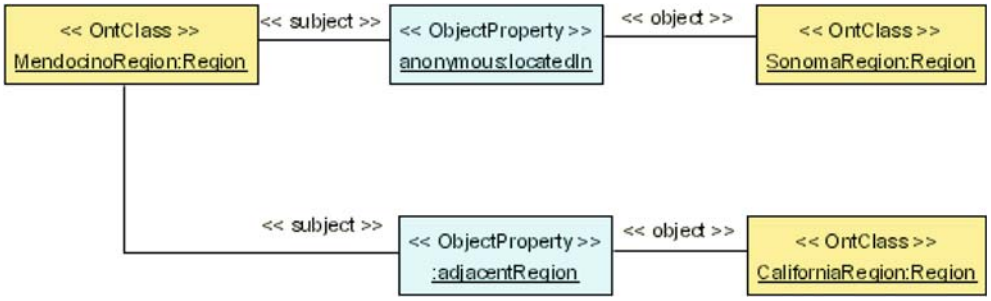

Fig. 4 OUP fully supports ontology body of knowledge (i.e., instances) through OUP statements: the Wine ontology example

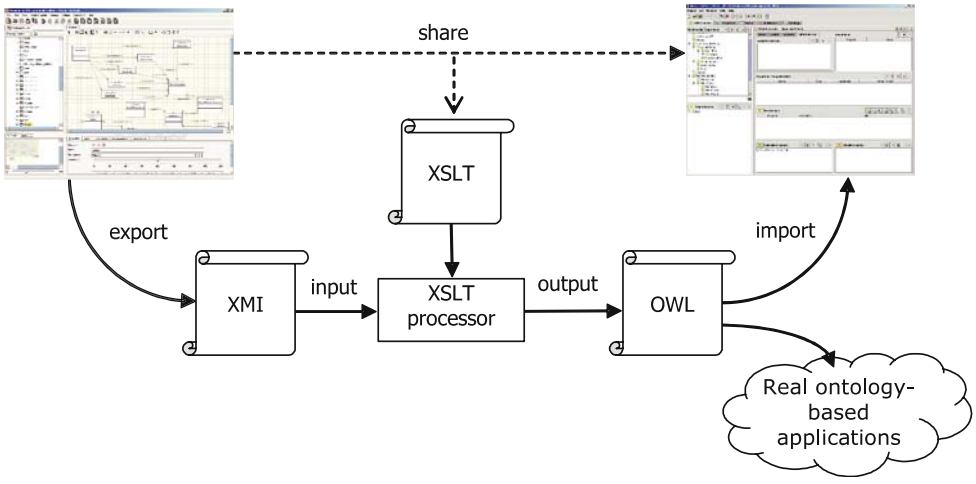

Fig. 5 The applied XSLT principle: an extension of present UML tools for ontology development

of the UML tool. Accordingly, we can use well-defined XML/XSLT procedure that is shown in Fig. 5.

A UML tool (e.g., Poseidon for UML) can export an XMI document that an XSLT processor (e.g., Xalan – http://xml.apache.org) can use as the 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), 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, instead we are able to use this ontology description as a final OWL ontology.

The XSLT, which we have implemented for mapping from XMI (OUP-based) format to the OWL XML description, contains a set of rules (i.e., 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 the 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.
- OUP, in some cases, uses more than one UML construct to model one OWL element. For example, to model *someValesFrom* restriction using OUP (see Fig. 3), we need three UML classes and three relations (i.e., one association and two dependencies). It is especially difficult because each UML construct has 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 used to check XMI documents. This is the only way to avoid generation of OWL ontologies with some incomplete parts (e.g., the attribute name of the property *owl:onProperty* in a class restriction is empty) if the input XMI document contains an incomplete UML model according to the OUP definition (e.g., there is no stereotyped dependency ≪ `onProperty`≫ between a restriction and a property).
- The XSLT must differentiate between classes that are defined in other classes (and can not be refer-

```
<xsl:template name="ObjectProperty">

    <xsl:variable name="range">
        <xsl:text>Class</xsl:text>
    </xsl:variable>

    <xsl:variable name="classID" select="./@xmi.id"/>

    <xsl:element name="owl:ObjectProperty">
        <xsl:attribute name="rdf:ID">
            <xsl:value-of select="./@name"/>
        </xsl:attribute>

        <xsl:call-template name="classDepedencyStereotype">
            <xsl:with-param name="stereotype">
                <xsl:text>equivalentProperty</xsl:text>
            </xsl:with-param>
        </xsl:call-template>

        <xsl:call-template name="taggedValues"/>

        <xsl:call-template name="generalization">
            <xsl:with-param name="generalizationKind">
                <xsl:text>rdfs:subPropertyOf</xsl:text>
            </xsl:with-param>
        </xsl:call-template>

        <xsl:call-template name="classDepedencyStereotype">
            <xsl:with-param name="stereotype">
                <xsl:text>inverseOf</xsl:text>
            </xsl:with-param>
        </xsl:call-template>

        <xsl:call-template name="attributeDomainRange"/>

        <xsl:call-template name="associationDomainRange">
            <xsl:with-param name="range" select="$range"/>
        </xsl:call-template>
    </xsl:element>

</xsl:template>
```

Listing 1 The XSLT's template that generates OWL object properties form an OUP model

enced from other classes using their ID), and classes that can be referred to using their ID. Accordingly, we included into OUP *odm.anonymous* tagged values that help us detect these two cases.

Taking into account previously presented facts, one can deduce that the developed XSLT is too large to be included in this paper. Therefore, we only give a part of it in Listing 1.

This listing illustrates the XSLT *ObjectProperty* template responsible for processing OUP's ≪`ObjectProperty`≫ stereotypes. The template is called from a template that matches XMI tags for the UML's class, but only when a matched class has attached the ≪`ObjectProperty`≫ stereotype. ≪`ObjectProperty`≫ outputs the `owl:ObjectProperty` XML tag, and calls templates responsible for transformation of the following elements: equivalent properties (dependency stereotype), attached tagged values that describe property type (i.e., symmetric, transitive, functional, and inverse functional), super properties (generalization), inverse properties, and property domain and range. In order to depict an output OWL document that we obtain as the XSLT's result, we give Listing 2.

Listing 2a gives OWL classes we have defined in Fig. 2. It is interesting to note how OUP's classes that have tagged value *odm.anonymous* are mapped into OWL (e.g., `WineDescriptor` has an equivalent anonymous class that is defined as a union of the `WineTaste` and `WineColor` classes). In Listing 2b, we show the OWL description for the `locatedIn` property, which has the `Region` class as its range, and both the `Region` and `Wine` classes as its domain. On the other hand, the `Wine` class additionally restricts this property using the OWL's *someValuesFrom* restriction. Since OUP has a full support for OWL's statements we are able to transform them into equivalent OWL's constructs (i.e., full individual descriptions). Listing 2c contains OWL instances defined as statements' parts in Fig. 4. This feature empowers our solution to generate both ontology armature [13] (classes, properties, etc) and ontology instances (body of knowledge) [8]. This feature is not supported in other MDA-based proposals for ontology development. Of course, we should note that Listing 2 is only a part of the OWL description of the Wine ontology that is obtained by performing the XSLT. In Sect. 5, we outline our first practical experience with this solution.

5 Experiences

We have already noted that the developed solution acts as an extension for standard UML tools, and thus enables us to create complete OWL ontologies without need to use ontology-specialized development tools. In order to accomplish a real practical use of the OUP and the developed XSLT, we should use 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.
- A convenient way to use tagged values and attach them to each UML element.
- Making relations between UML concepts, as those shown in Fig. 2. We especially emphasize the importance of relations (e.g., dependency) between a UML class and a UML object. This kind of relation is regular in the UML syntax, and can be represented on

```
<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)

Listing 2 Resulting OWL description: **a** classes generated for the OUP model from Fig. 2; **b** Object property OWL descriptors for the model from Fig. 3 **c** OWL statements generated from Fig. 4

class diagrams (that are also called static structure diagrams in the UML specification [38]).
- The XMI standard for serialization of UML models.

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 the 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 (http://mdr.netbeans.org) repository for MOF-compliant metamodel storing, and the MOF definition itself. This is important feature because the usage of model repositories enables us to benefit from all MDA's advantages [6]. From the historical 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's UML backend for importing UML models.

The second important decision is how to generate 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 decide to generate OWL ontologies in the fashion similar to Protégé's OWL plugin. Hence, we have managed to provide an additional way to import Poseidon's models into Protégé through the OWL. Of course, since Protégé has more advanced features for ontology development, an OUP-defined ontology can be further improved and refined.

We have tested our solution on the well-known example of the Wine ontology [28]. Firstly, we represented this ontology in Poseidon using OUP. Parts of this ontology are used in Sect. 4 in order to illustrate OUP (e.g., Figs. 2, 3). Then, we exported this extended UML into XMI, and 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 Fig. 6.

We have to admit that we have found certain difference between OWL generated by the XSLT and OWL produced by Protégé. That difference was detected in representation of OWL's individuals. To represent individuals Protégé uses `owl:Thing` with the attribute `rdf:type` that refers to its type (i.e., its OWL class).

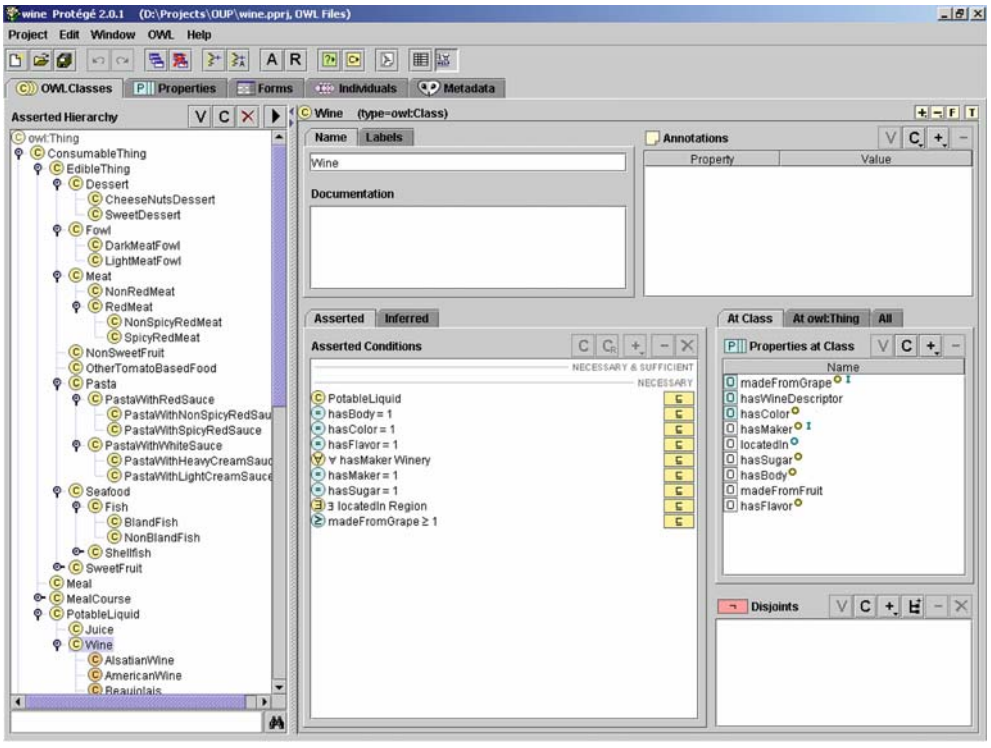

Fig. 6 An example of generated OWL ontology from the OUP, and imported into Protégé: Wine ontology

For example, `Red` is 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 by a tag that has the same name as the name of 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.

The current XSLT version has a limitation, since it does not support packages (i.e., the OUP ontology). This means, it is unable to produce more than one OWL document (i.e., ontology). Actually, the 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 by using some non-standard XSLT primitives (i.e., XSLT extensions) that enable producing multiple documents from one source XML document (e.g., SAXON XSLT processor and its XSLT extensions).

We have so far developed two ontologies using the OUP that we later transformed in OWL using the XSLT. These two ontologies are: the ontology of saints and philosophers and the Petri net ontology. The first ontology was developed using Porphyry's tree method – starting from Porphyry's tree schema, performing forward classification, and establishing the class hierarchy. This ontology has a theological and philosophical character, but also contains a multimedia part that semantically annotates a collection of pictures collection (icons) [11]. The Petri net ontology was developed in order to provide the Semantic Web support for Petri nets [19].

6 Discussion

In order to further analyze the proposed solution, we compare our Ontology UML Profile with the UML Profile proposed in [1] (the most similar approach). We also indicate the status of the implemented transformations in context of the ongoing OMG initiative for ontology development.

6.1 Ontology UML profile

We can say that both ODM and OUP used in this paper are mostly related to the work of Baclawski et al. [1]. Here, we give a difference between our OUP and Baclawski's UML profile on the example of a restriction. In Fig. 7a, we show how restriction on property `locat-`

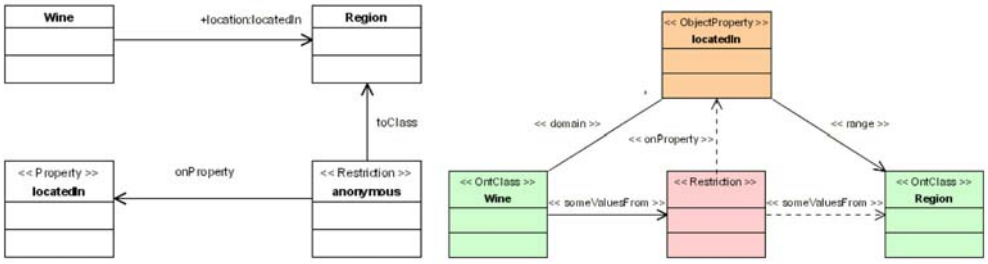

Fig. 7 Definition of *locatedIn* property for the class Wine represented in: **a** Becklawski's UML profile, **b** Ontology UML Profile

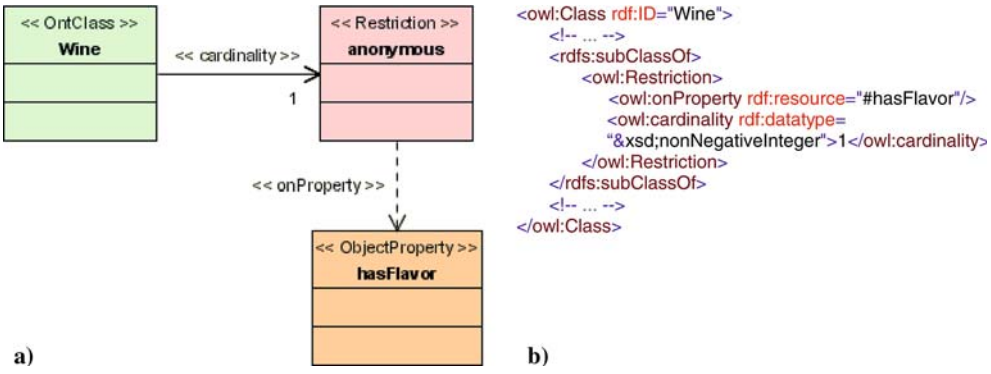

Fig. 8 The *Wine* class and cardinality definition for the property *hasFlavor*: **a** OUP definition **b** OWL definition

edIn is defined using the Baclawski's UML profile and in Fig. 7b, we present how it would be defined by using OUP (also shown in Fig. 3). This example is taken from the well-known Wine ontology.

In OUP, we have «domain» association stereotype to attach domain classes to a property. For property range, we use a directed association stereotype «range», while in Baclawski's UML profile, they use an unnamed «Restriction» stereotype that has two associations: toClass and onProperty. In OUP, we use the «Restriction» stereotype additionally to refine restriction on a property for some class. In that case, we have an association (e.g., stereotype «someValuesFrom») from a class to an unnamed «Restriction», and two stereotyped dependencies from «Restriction»– «on Property», and e.g., «allValuesFrom» (but, here also can be used stereotypes «hasValue», «someValuseFrom»). It is important to note that having this «Restriction» construct in OUP does not mean adding a class into property domain. Actually, it is mapped as a super class for a given class in the way shown in Listing 2b.

In addition, we need a way to change parent class multiplicity for a property in its inherited classes. This is not precisely defined in Baclawski's UML Profile. In OUP, we use another association stereotype «cardinality» between an «OntClass», which we additionally want to restrict, and the stereotype «Restriction». Multiplicity defined on the «Restriction» side is a new cardinality. Further, «Restriction» is related with a property through a dependency stereotype «onProperty». Fig. 8a illustrates an example of a cardinality restriction for the class Wine and on the property hasFlavor. Also it gives an equivalent OWL definition (see Fig. 8b).

Of course, one important question naturally rises: Why do we need a UML Profile for ontology modeling? Having in mind the fact that ontologies are a very dynamic category evolving continuously, it is very important for developers to have an intuitive way to present ontology artifacts. According to one of the most comprehensive and referred ontology tool surveys [12], it is obvious that many ontology tools take into account this fact by providing different graphical notations to represent ontologies. For example, Protégé, the most popular ontology editor, has some plug-ins for graphical representation of ontologies (e.g., OntoViz, TGViz). The same practice is continued in Protégé's support for OWL where there is a plug-in called ezOWL (http://iweb.etri.re.kr/ezowl) that uses a UML-like graphical notation for OWL ontologies. However, it is important to note that the use of some standard graphical software notations (e.g., ER diagrams and UML) is very common in many ontology tools (DUET, Visual Ontology Modeler, ICOM, KBE, etc.) [12]. On the other hand, one more important question is: Do we really need a UML Profile

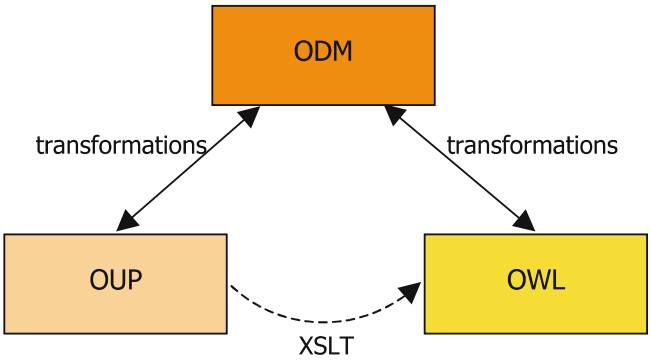

Fig. 9 Relations between implemented solution and recommended transformations in OMG's Request for Proposals

for ontologies when we already have Protégé? Although this question may sound quite reasonable, the fact is that UML tools such as Rational Rose have many advantages: availability of literature, courses, examples, know-how tutorials, industrial support, etc. Of course, one can strength Protégé to have more advanced features than some UML tools, but using well-known modeling standards is still an argument for having UML profiles for ontologies.

6.2 Transformation

One very important remark is that the proposed XSLT transformation, in fact, is not a part of OMG's RFP for Ontology definition metamodel [36]. This document presumes transformations between ODM and OUP, as well as transformations between ODM and OWL (see Fig. 9). 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 the XML: ODM uses the XMI format – a MOF-defined metamodel, OUP uses the UML XMI format, and OWL has an XML-based representation. Our transformation from OUP to OWL is 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. Actually, MDA is still supported only with UML tools designed for UML modeling at the M1 layer of the MDA [20]. Development of these ODM↔OUP and ODM↔OWL transformations is currently our primary activity.

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 ODM, and from ODM to a new language. In the case we want to support transformations between *N* different languages (like the OUP and OWL are), then it is necessary to implement *2N* transformations. But, when 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). In this way, we are capable to prevent transformation of an invalid ontology or to alert when an ontology is inconsistent. This is similar to relations between EBNF notation most commonly used for defining programming grammars and concrete programming languages [5]. In terms of EBNF, we can check a validity of a program written in a programming language (e.g., Java) against the EBNF-based grammar of that programming language when parsing the program. Trying to make relations with EBNF we can say that a metamodel (e.g., ODM) is equivalent to an EBNF grammar, while a concrete model (e.g., ontology) is equivalent to a program.

Our transformation mechanism is similar to the ideas given in Bézivin's paper [5] that propose metamodel-based model transformations and implementation of transformations using the XSLT. Here, we have different metamodels (i.e., OWL, OUP metamodel). However, ODM serves as an integration point that decreases the number of needed transformations. Also, we can prove usefulness of having a central metamodel for some area, which, in this case, is ODM.

Note that the model sharing principle illustrated in Fig. 9 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 generic model for frame representation and, in fact, generic interface to different frame representation systems (FRS) [24]. 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 [9].

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) [14] compliant code, and thus obtain a possibility to incorporate a programming logic into Java applications. The JMI Specification defines a Java-based 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 both are defined using MOF (UOP 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.

7 Conclusions

In this paper, we have shown a practical realization of the UML tool extension for development of real Semantic Web ontologies. This extension is implemented in XSLT. Actually, the implemented XSLT transforms an ontology described by the Ontology UML Profile (XMI) into the W3C's Semantic Web language OWL (XML). The used Ontology UML Profile, and the developed XSLT are part of our efforts to develop a software engineering platform for ontology development, which will be in accordance with the OMG's RFP for Ontology Definition Metamodel [36]. Unlike previous similar solutions, our implementation produces an ontology that does not need to be additionally refined by ontology development tools. We recommend (but, do not limit because the solution is based on the UML XMI standard) using this XSLT together with the Poseidon for UML, since it supports all UML concepts necessary for the Ontology UML Profile. Generated OWL ontologies have a similar look to ontologies produced by Protégé's OWL plugin.

We hope that this solution can be useful to all software engineering practitioners who participate in an ontology development process. Using the well-known UML syntax, the practitioners do not need to learn how to use ontology tools. Also, we hope that this work can be useful practical contribution to the OMG's efforts in finding a suitable MDA-based technique for the Semantic Web ontologies that will bring ontology development process closer to software engineers.

In the future, we are planning to improve the current implementation, so that it can support development of multiple ontologies (using UML's packages), and show how the Ontology UML Profile can be used for modular ontology development (on the example of the Petri net ontology and Petri net dialects). Also, we will finish our current work aimed at providing support for transformations between the Ontology UML Profile (UML XMI-based) and the Ontology Definition Metamodel (MOF XMI-based), as well as between the OWL and the Ontology Definition Metamodel. In this way, we will have an entire metamodeling platform compliant to the OMG's ontology initiative.

References

1. Baclawski, K., Kokar, M., Kogut, P., Hart, L., Smith, J., Letkowski, J., Emery, P.: Extending the Unified Modeling Language for ontology development. Int J Softw. Syst. Modeling **1**(2), 142–156 (2002)
2. Baclawski, K., Kokar, M., Smith, J., Wallace, E., Letkowski, J., Koethe, M., Kogut, P.: 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)
3. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L.: OWL Web Ontology Language Reference. W3C recommendation http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (2004)
4. Berners-Lee, T.: Weaving the Web. Orion Business Books, London (1999)
5. 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, pp. 350–355 (2001)
6. Bock, C.: UML without pictures. IEEE Software **20**(5), 33–35 (2003)

7. Brickley, D., Guha, R.: (eds.) Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation http://www.w3.org/TR/2000/CR-rdf-schema-20000327 (2000)
8. Chandrasekaran, B., Josephson, J., Benjamins, R.: What are ontologies, and why do we need them?. IEEE Intell. Syst. **14**(1), 20–26 (1999)
9. Chaudhri, V., Farquhar, A., Fikes, R., Karp, P., Rice, J.: OKBC: A programmatic foundation for knowledge base interoperability. In: Proceedings of the 15th National Conference on Artificial Intelligence, Madison, Wisconsin, USA, pp. 600–607 (1998)
10. Cranefield, S.: Networked knowledge representation and Exchange using UML and RDF. J. Digi. inf. 1(8) http://jodi.ecs.soton.ac.uk (2001)
11. Damjanović, V.: Semantic Web, Ontologies, and Agents. Honors degree thesis, University of Belgrade, Serbia and Montenegro (2003)
12. Denny, M.: Ontology Tools Survey, Revisited http://www.xml.com/lpt/a/2004/07/14/onto.html (2004)
13. Devedžić, V.: Understanding Ontological Engineering. Communications of the ACM **45**(4), 136–144 (2002)
14. Dirckze, R. (ed.) Java Metadata Interface (JMI) Specification Version 1.0 http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html (2002)
15. Djurić, D., Gašević, D., Devedžić, V.: Ontology modeling and MDA. J. Object Techno. **4**(1) 109–129 (2005)
16. Duddy, K.: UML2 must enable a family of languages. Commun. ACM **45**(11), 73–75 (2002)
17. Falkovych, K., Sabou, M., Stuckenschmidt, H.: UML for the semantic web: Transformation-based approaches. In: Omelayenko B, Klein M (eds.) Knowledge Transformation for the Semantic Web. Frontiers in Artificial Intelligence and Applications 95, IOS Press, Amsterdam, The Netherlands, pp. 92–106 (2003)
18. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D., Patel-Schneider, P.: OIL: An Ontology Infrastructure for the semantic web. IEEE Intell. Syst. **16**(2), 38–45 (2001)
19. Gašević, D., Devedžić, V.: Reusing petri nets through the semantic web. In: Proceedings of the 1st European Semantic Web Symposium, Heraklion, Greece, 2004. LNCS 3053. Springer, Berlin Heidelberg New York, pp 284–298 (2004)
20. Gašević, D., Damjanović, V., Devedžić, V.: Analysis of the MDA standards in ontological engineering. In: Proceedings of the 6th International Conference of Information Technology, Bhubaneswar, India, pp. 193–196 (2003)
21. Gómez-Pérez, A., Corcho, O.: Ontology languages for the semantic web. IEEE Intell. Syst. **17**(1), 54–60 (2002)
22. Gruber, T.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
23. Juerjens, J.: Secure Systems Development with UML. Springer, Berlin Heidelberg New York (2003)
24. Karp, P., Myers, K., Gruber, T.: The generic frame protocol. In: Proceedings of the 1995 International Joint Conference on Artificial Intelligence, Monreal, Canada. pp. 768–774 (1995)
25. Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M., Smith, J.: UML for ontology development. Knowl. Eng. Rev. **17**(1), 61–64 (2002)
26. McGuinness, D., Fikes, R., Hendler, J., Stein, L.: DAML+OIL: An Ontology Language for the Semantic Web. IEEE Intell. Syst. **17**(5), 72–80 (2002)
27. 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 (2003)
28. Noy, N.F., McGuinness, D.: Ontology Development 101: A Guide to Creating Your First Ontology. TR SMI-2001-0880, Knowledge Systems Laboratory, Stanford University (2001)
29. Noy, N.F., Fergerson, R., Musen, M.: The knowledge model of Protégé-2000: combining interoperability and flexibility. In: Proceedings of the 12th International Conference on Knowledge Acquisition, Modeling and Management, Juan-les-Pins, France, pp. 17–32 (2000)
30. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R., Musen, M.: Creating Semantic Web Contents with Protégé-2000. IEEE Intell. Syst. **16**(2), 60–71 (2001)
31. Ontology Definition Metamodel Preliminary Revised Submission to OMG RFP ad/2003-03-40, Volume 1 http://codip.grci.com/odm/draft (2004)
32. Ontology Definition Metamodel, DSTC Initial Submission. OMG Document ad/2003-08-01 http://www.omg.org/cgi-bin/doc?ad/03-08-01 (2003)
33. Ontology Definition Metamodel, Gentleware Initial Submission. OMG Document ad/03-08-09 http://www.omg.org/cgi-bin/doc?ad/03-08-09 (2003)
34. Ontology Definition Metamodel, IBM Initial Submission. OMG Document ad/03-07-02 http://www.omg.org/cgi-bin/doc?ad/03-07-02 (2003)
35. OMG 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 (2002)
36. OMG Ontology Definition Metamodel Request for Proposal. OMG Document: ad/2003-03-40 http://www.omg.org/cgi-bin/doc?ad/2003-03-40 (2003)
37. Ontology Definition Metamodel, Sandpiper Software Inc and KSL Initial Submission. OMG Document ad/03-08-06 http://www.omg.org/cgi-bin/doc?ad/03-08-06 (2003)
38. OMG Unified Modeling Language Specification v1.5. OMG Document formal/03-03-01 http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.zip (2003)
39. OMG XMI Specification, v1.2. OMG Document formal/02-01-01 http://www.omg.org/cgi-bin/doc?formal/2002-01-01 (2002)
40. Seidewitz, E.: What Models Mean. IEEE Software **20**(5), 26–32 (2003)
41. Selic, B.: The pragmatics of model-driven Development. IEEE Software **20**(5), 19–25 (2003)
42. Sigel, J.: Developing in OMG's Model-Driven Architecture, Revision 2.6. OMG's White Paper ftp://ftp.omg.org/pub/docs/-omg/01-12-01.pdf (2001)