CHAPTER 1

MDA-BASED ONTOLOGICAL ENGINEERING

[1]DRAGAN DJURIĆ, [1,2]DRAGAN GAŠEVIĆ,
[1]VIOLETA DAMJANOVIĆ and [1]VLADAN DEVEDŽIĆ

[1]GOOD OLD AI Research Group, FON, University of Belgrade,
POB 52, Jove Ilića 154, 11000 Belgrade, Serbia and Montenegro
[2]School of Interactive Arts and Technology, Simon Fraser University Surrey
2400 Central City, 10153 King George Hwy.Surrey, BC V3T 2W1, Canada
E-mail: dragandj@gmail.com, gasevic@yahoo.com,
vdamjanovic@gmail.com, devedzic@fon.bg.ac.yu

The chapter presents a concept of approaching two ongoing technologies, ontological engineering and OMG's Model Driven Architecture (MDA), which are developing in parallel, but by different communities. Our main intention is to show recent efforts that pursuing to provide software engineers to use and develop ontologies. Many authors have so far stressed this problem and have proposed several solutions and some of them are analyzed in this chapter. The result of these efforts is the recent OMG's initiative for defining an ontology development platform. The ontology platform should be defined using MDA-based standards and it should consist of: Ontology Definition Metamodel, Ontology UML Profile, and a set of transformations. We depict our proposal for an MDA-based ontology development platform in order to illustrate this OMG's effort as it is in a very initial stage and a formal recommendation has not been adopted yet.

1. Introduction

The Semantic Web and its eXtensible Markup Language (XML) based languages are the main directions of the future Web development. Domain ontologies [30] are the most important part of the Semantic Web applications. They are formal organization of domain knowledge, and in that way enable knowledge sharing between different knowledge-base applications. Artificial intelligence (AI) techniques are used for ontology creation, but those techniques are more related to research laboratories, and they are unknown to wider software engineering population.

The integration of the ongoing software engineering efforts with the concept of the Semantic Web is not a new idea [13, 40]. The main question is how to develop the Semantic Web ontologies using well-accepted software engineering languages and techniques in order to provide the wider practitioner population to develop and use 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. Hence, we can only use UML in initial phases of an ontology development. We believe that these limitations can be overcame using UML's extensions (i.e., UML profiles) [19], 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 [50]. The most important direction toward this goal is the Special Interest Group (SIG) within Object Modeling Group (OMG) that will converge many different proposals regarding this problem [44]. The result of this effort should be a standard language (i.e., metamodel) based on the MDA standards [42] and the W3C's Web Ontology Language (OWL) recommendation [6].

The next section contains an overview of the ontologies and the Semantic Web, while Sec. 3 describes the Semantic Web languages and OWL. Section 4 defines OMG's MDA initiative and related concepts: Meta-Object Facility, UML Profiles, and XML Metadata Interchange (XMI). In Sec. 5, we give an overview of current work using MDA-based solutions for ontology development. In Sec. 6, we give a framework for the ontology language metamodel in the context of the OMG's effort. Section 8 shows the ontology metamodel definition in detail while Sec. 9 gives description of Ontology UML Profile. The last section contains the final conclusions. This work is a part of the effort of the GOOD OLD AI research group (*http://goodoldai.org.yu*) in developing AIR - a platform for building intelligent information systems.

2. An overview of the Ontologies and the Semantic Web

Ontologies have been around for quite some time now. Since early 1990s researchers in the domain of artificial intelligence and knowledge representation have studied ontologies as means for knowledge sharing and reuse among knowledge-based systems. However, even an early survey of the field of ontologies [24] has identified a number of application classes that benefit to a large extent from utilizing ontologies although some of them are not necessarily

knowledge-based systems in the traditional sense. Some of the application classes it mentioned include natural language processing, library science, intelligent information retrieval (especially from the Internet), virtual organizations, and simulation and modeling. Later on, researchers have recognized explicitly that ontologies are not just for knowledge-based systems, but for all software systems – all software needs models of the world, hence can make use of ontologies at design time [11]. Nowadays, ontologies and ontological engineering span such diverse fields as qualitative modeling, language engineering, database design, information retrieval and extraction, knowledge management and organization, ontology-enhanced search, possibly the largest one, e-commerce (e.g., Amazon.com, Yahoo Shopping, etc.), and configuration [41].

2.1. *Definitions and background*

There are at least a dozen definitions of ontologies in the literature. A recent one says that ontology provides the basic structure or armature around which a knowledge base can be built [52]. Another one specifies that ontology should provide a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic or service [32]. Although informal, these definitions capture the central idea of ontologies – they are structured depictions or models of known (and accepted) facts about some topics. Ontologies appear most effective when the semantic distinctions that humans take for granted are crucial to the application's purpose [15].

Each ontology provides the vocabulary (or names) for referring to the terms in a subject area, as well as the logical statements that describe what the terms are, how they are related to each other, how they can or cannot be related to each other, as well as rules for combining terms and relations to define extensions to the vocabulary. Hence, ontologies represent a common machine-level understanding of topics that can be communicated between users and applications, i.e., domain semantics independent of reader and context. For a more recent comprehensive discussion of ontologies, see [36].

2.2. *Semantic Web*

One of the central roles of ontologies is to establish further levels of interoperability, i.e., semantic interoperability, between agents and applications on the emerging Semantic Web [8], as well as to add a further representation and

inference layer on top of the Web's current layers [14, 32]. When put on the Web, ontologies specify standard terms and machine-readable definitions. The Semantic Web is based on the idea of numerous ontologies providing vocabularies, definitions, and constraints that information resources, agents, and Web-based applications can commit to in order to reuse data and knowledge effectively [31]. This way, ontology conveys the same meaning of its terms to any two or more sources that commit to it. Any source, agent, or application can commit to any ontology or create a new one. Thus, the Semantic Web is essentially a distributed approach to creating standard vocabularies.

2.3. *Ontological engineering*

The engineering part of developing ontologies comprises a complex set of activities that are conducted during conceptualization, design, implementation and deployment of ontologies. Ontological engineering covers a whole range of topics and issues, such as the basics (philosophical and metaphysical issues and knowledge representation formalisms), methodology of ontology development, recent Web technologies such as XML [7] and its relatives [38], business process modeling, commonsense knowledge, systematization of domain knowledge, Internet information retrieval, standardization, evaluation, ontology integration with agents and applications, and many more [16]. It also gives us design rationale of a knowledge base, helps us define the essential concepts of the world of interest, allows for a more disciplined design of a knowledge base, and enables us to accumulate the knowledge about it. The disciplines tightly interwoven with ontological engineering include modeling, metamodeling, and numerous fields of software engineering.

2.4. *Ontology building tools*

An important aspect of building ontologies is the use of specific software tools that enable ontology conceptualization, representation, construction, and use. There are a number of such tools today. Most of them have resulted from efforts of research groups and university labs, and are currently free. However, these tools can differ to a large extent in terms of support they provide to the ontology development process, the format(s) used for storing ontologies, the number of format converters supported for translating ontologies to/from other formats, the way(s) other applications can interoperate with ontology tools, the tool stability and maturity, support for querying information about an ontology, and so on [29].

3. An overview of the Semantic Web Tools and Languages

There were several efforts so far to develop a comprehensive classification of ontology development tools, as well as to compare and evaluate a number of different tools. The most comprehensive among such approaches to date is the one proposed by OntoWeb Consortium [29]. The approach starts from grouping all ontology-based software tools into the following large categories:

- ontology development tools – the tools, environments and suites that can be used for building a new ontology from scratch or reusing existing ontologies;
- ontology merge and integration tools – the tools helping to solve the problem of merging or integrating different ontologies on the same domain;
- ontology evaluation tools – support tools that enable getting insight into the level of quality of ontologies and their related technologies;
- ontology-based annotation tools – the tools enabling the users to insert ontology-based markups in Web pages;
- ontology storage and querying tools - the tools that allow using and querying ontologies easily; and
- ontology learning tools - the tools used to (semi) automatically derive ontologies from natural language texts.

A similar, though much more narrowly focused study by M. Denny, covered ontology editors only [15]. Ontology browsers without an editing focus and other types of ontology building tools were not included. The study was still very useful because it helped identify a cross-section of ontology editing tools.

Another group of comparative studies is focused on ontology development languages only. A good example coming from an academic environment is the study of languages for the Semantic Web [28]. The study has identified three levels of abstraction of such languages and has included only the languages based on XML technologies.

We propose a suitable, practically oriented, and simple framework/hierarchy that can be used for an easy, yet very informative categorization of ontology development tools. It is drawn based on informal criteria of the tools' sophistication and usability. Despite the fact that these may appear as rather subjective criteria, they do allow for a rough hierarchical categorization of all currently available ontology development tools. The framework is characterized by:

- a wider focus than that of ontology editors alone, used in [15];
- yet, a more narrow focus than that of covering all ontology-related tools as in [29] – our framework concentrates on ontology development tools only;

- ontology development languages themselves are included, although much less formally than in [28, 48];
- ontology learning tools are included, since ontology learning is also a way of *building* ontologies.

Figure 1 describes the framework/hierarchy graphically.

Ontology learning tools	Tools employing machine learning
Ontology-development environments	Integrated graphical tools
Ontology-representation languages **(The Semantic Web languages)**	Languages of different expressive power and based on different representation paradigms (regardless of the underlying technology)
XML/RDF	XML/XMLS, RDF/RDFS and the corresponding development tools

Fig. 1. Hierarchy of ontology development tools.

3.1. *The Semantic Web languages*

Common data interoperability in present applications is best achieved by using XML. XML is a meta-language used to define other languages. It describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them [9]. XML defines neither the tags nor grammar, which makes it completely extensible. It only requires that document must be well-formed in a tree structure, so it could be parsed by standard XML tools. Hence, one can view XML technologies and languages, as well as their corresponding development tools, as constituting the core of ontology development tools. Ontologies represent semantics and meanings of topics and subject areas in a declarative form. XML syntax is suitable for ontology representation because it is human readable, simple to parse, well defined, and widely used. Fundamental XML-based languages – XML itself, XMLS, Resource Description Framework (RDF) and RDF Schema (RDFS) – can express some semantics themselves.

Whereas in pure HTML the tags are fixed, in XML they are arbitrary and are described in a Document Type Definition (DTD) or in an XMLS document. Having custom tags in a document adds context and gives meaning to data and

let people meaningfully annotate text [6]. Using XMLS to prescribe the data structure, XML can encode all kinds of data that is exchanged between computers. This brings an extensible and easy-to-use syntax for describing Web data, though just a minimum semantics. With respect to the Semantic Web technology, it is important to stress a role of an XML Metamodel Interchange (XMI) as a standard for stream-based model interchange. The main purpose of XMI [46] is to enable easy interchange of metadata between modeling tools (based on the Object Management Group (OMG) Unified Modeling Language (UML)) and between tools and metadata repositories (OMG Meta Object Facility (MOF)) in distributed heterogeneous environments. XMI integrates three key industry standards:

(i)**XML** - a W3C standard;
(ii)**UML** - an OMG modeling standard; and
(iii)**MOF** - Meta Object Facility and OMG modeling and metadata repository standard.

The integration of these three standards into XMI marries the best of OMG and W3C metadata and modeling XMI technologies allowing developers of distributed systems to share object models and other metadata over the Internet. XMI standardizes the exchange of metamodels, models, as well as object instances between applications [46].

Apart from the XML, there are other languages attempt to achieve semantic interoperability. Such languages are Ontology Interchange Language (Ontology Inference Layer) (OIL), DARPA Agent Markup Language (DAML+OIL), RDF, RDFS, and Web Ontology Language (OWL).

OIL is a proposal for a joint standard for describing and exchanging ontologies. OIL permits semantic interoperability between web resources. OIL is not just another new language but reflects a certain consensus among the specialists in the areas such as description logic (DL) and frame-based systems. OIL is a significant source of inspiration for the ontology language DAML+OIL [21].

DAML+OIL is an ontology language specifically designed for use on the Web, as a joint effort to create a standard language for the Semantic Web. DAML+OIL uses existing standards (XML and RDF) adding the familiar ontological primitives of object-oriented and frame-based systems, and the formal rigor of a very expressive DL [34]. DAML+OIL is built on top of W3C standards such as RDF and RDFS, and extends these languages with richer modeling primitives [33].

RDF and RDFS cannot be considered as ontology specification languages, but rather as general languages for the description of metadata on the Web [12].

It is important to stress that they are a W3C standard for the Semantic Web. RDF is a framework for representing metadata, i.e., a model for representing data about resources on the Web. Each RDF description is basically a list of *object* (resource) - *attribute* (property) - *value* (resource or free text) triples, i.e., statements. This RDF data model is equivalent to the semantic network formalisms, which consist of three object types: properties, resources, and statements.

RDFS are used to define an RDF document vocabulary (domain-specific properties and classes of resources to which those properties can be applied), and are referred to in RDF documents through namespaces. It is important to stress that RDFS uses modeling primitives like *class*, *subclass-of*, *property*, *domain* and the like, with much higher expressive power than those used in XMLS. These allow for specifying higher-level semantics and can be used for basic ontology modeling.

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the WWW. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL Web Ontology Language. OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF and RDFS by providing additional vocabulary along with a formal semantics. OWL has three variants [6]:

- OWL Lite is intended mostly to support classification hierarchy and simple constraint features. It is a good starting point for tool builders. OWL Lite can be useful in migrations of existing taxonomies to OWL.
- OWL DL enables maximum expressiveness and guarantees computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time). It includes all OWL Full constructs, and appends some constraints. The most significant constraints are that a class cannot be an individual or a property, or that a property cannot be an individual or a class. OWL DL has a good formal background since it is based on description logics.
- OWL Full provides maximum expressiveness and syntactic independence of RDF, but doesn't provide any computational guarantees. The main characteristic of OWL Full in comparison to OWL DL and OWL Lite is that one class, which is, by definition, a collection of individuals, can be an individual itself, like in RDF(S). This approach can lead to models that need infinite time to compute.

OWL Full is an extension of OWL DL, which is an extension of OWL Lite, thus every OWL Lite ontology is OWL DL and OWL Full ontology and every OWL DL ontology is OWL Full ontology. The place of OWL in described architecture is shown in Fig. 2.

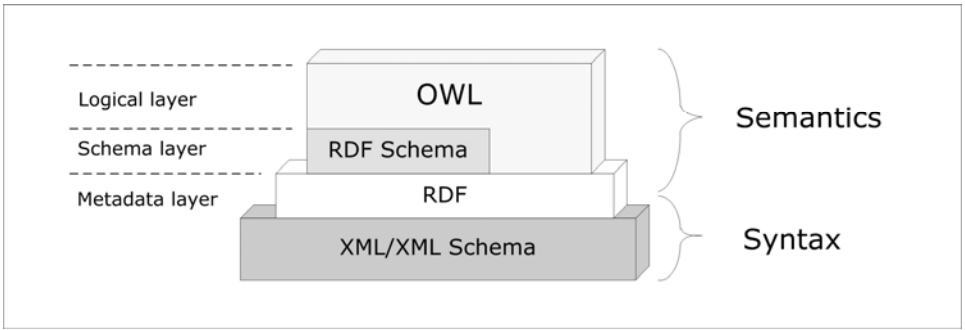

Fig. 2. OWL in the Semantic Web architecture.

Since the World Wide Web is almost unconstrained, OWL must support open world assumption and allow importing and mixing various ontologies. Some of them may be even contradictory, but new information can never retract existing information, it can be only added to it.

4. An Overview of Model Driven Architecture and Meta-Object Facility

If we look back to the history of software development, we can see a notable increase of models abstraction. Modeling becomes more and more separate from underlying platforms, making models of real world more reusable and easy to create by domain experts, requiring less knowledge of specific computer systems. This places software modeling closer to knowledge acquisition in knowledge engineering and vice versa. Current stage in that evolution is OMG's Model Driven Architecture [42].

4.1 *MDA basics*

MDA defines three viewpoints (levels of abstraction) from which a system can be seen. From a selected viewpoint, a representation of a given system (viewpoint model) can be defined. These models are (each corresponding to the viewpoint with the same name): *Computation Independent Model* (CIM),

Platform Independent Model (PIM) and *Platform Specific Model* (PSM). CIM is a view of a system that does not show the details of a system structure. In software engineering, it is also known as a domain model, which is specified by domain experts. It is similar to the concept of ontology. PIM is the model that is computation dependent, but it is not aware of specific computer platform details. In other words, it is targeted for technology-neutral virtual machine. Specification of complete computer system is completed with PSM. The goal is to move human work from PSM towards CIM and PIM and let the specific platform detail implementations be generated as much as possible by automated tools which will do the transformation from PIM to PSM.

All metamodels, standard or custom, defined by MOF are positioned at the M2 layer. One of these is UML, a graphical modeling language for specifying, visualizing and documenting software systems. With UML profiles, basic UML concepts (Class, Association, etc.) can be extended with new concepts (stereotypes) and adapted to specific modeling needs. The models of the real world, represented by concepts defined in the corresponding metamodel at M2 layer (e.g. UML metamodel) are at M1 layer. Finally, at M0 layer are instances of concepts modeled at M1 layer. An example would be: MOF Class (at M3) is used to define UML Class (M2), which is used to define real-world describing concept, class Person (M1) that can have instances: Tom, Dick, Harry (M0).

Another standard that this architecture is based on is XMI, a standard that defines mapping from MOF-defined metamodels to XML documents and Schemas. XML, which is well-supported in various software tools, gives XMI strength to enable sharing of meta-metamodel, metamodels and models.

Present software tools support for MDA is concentrated primarily on UML as a graphical notation, with no concern of metamodeling layers [26]. UML CASE tools (e.g. Rational Rose, Borland Together, Magic Draw, Poseidon for UML, etc.) have good support for modeling at M1 layer and for code generation in certain programming languages. Using appropriate UML profile they can generate databases, XML Schemas, EJBs etc. But, they lack support for M2 and M3 layers as well as a unified serialization to XMI. It is expected from future tools to support UML 2, which will enable common XMI representation of UML models, and MOF-compliant model repositories at M2 and M3 layers; all this will provide a good support for metamodeling.

4.2. *Modeling: Instance layers versus ontological layers*

MOF is a self-defined language intended for defining metamodels. In term 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 [49]. Examples of the MDA's metamodels are UML and CWM. The MDA's metamodel layer is usually marked as M2. At this layer, we can define a new metamodel (e.g. modeling language) that would cover some specific application domains (e.g. ontology development). The next layer is the model layer (M1) – a layer where we develop real-world models (or domain models). In terms of the UML models that means creating classes, their relations, states, etc. This layered architecture, also shown in Fig. 3 is often difficult to understand for less experienced modelers, so we should explain the bottom-most layer, the instance layer (M0) in more depth. There are two different approaches about this question, and we note both of them:

(i) The instance layer contains instances of the concepts defined at model (M1) layer (e.g. objects in programming languages).
(ii) The instance layer contains things from our reality – concrete (e.g. Mark is instance of the Person class, Lassie is a instance of the Dog class, etc.) and abstract (e.g. UML's classes – Dog, Person, etc.) [3].

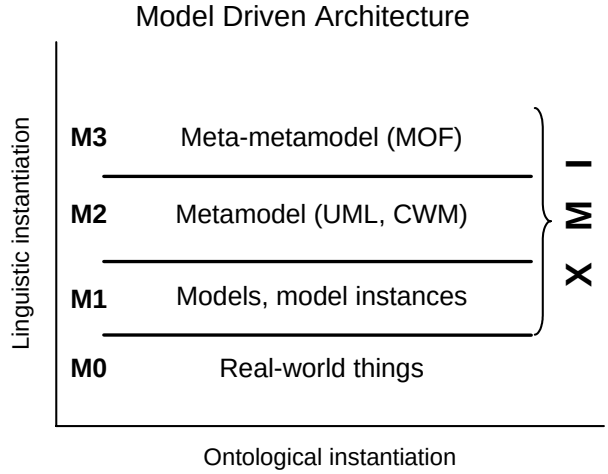

Fig. 3. The four-layer Model Driven Architecture and its orthogonal instance-of relations: linguistics and ontological.

In this chapter we advocate the second approach, but we should give a more details about its impact on UML. In UML, both classes and objects are at the same layer (model layer) in the MDA four-layer architecture. Actually, MDA's layers are called linguistic layers. On the other side, 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.

4.3. *Specific MDA metamodels and UML profiles*

One possible solution for using MDA capacities in some specific domains is to develop a metamodel, which would be able to model relevant domain concepts. That means, creating a domain language (i.e., metamodel) using metamodeling, and these languages are 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 so we try to use well-developed tools. Practically, present software tools do not implement many of the MDA basic concepts. However, most of these applications, currently primarily oriented toward UML and the M1 (i.e., model) layer [26]. Generally, UML itself is a MOF-defined general-purpose language (i.e., metamodel) that contains a set of core primitives. The problem of tools can be overcome using UML Profiles – a way for adapting UML for specific purposes. UML Profiles extend the UML metamodel with application-specific primitives (through stereotypes, tagged values, and constraints), and hence these primitives can be used as the regular UML concepts. Having understood UML Profiles in this way one can count UML as a family of languages [19].

A very important question is about the palace of UML Profiles in the MDA's 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, 2] where UML Profiles are placed at both the metamodel layer (M2) and the model layer (M1).

5. Current Trends: Using UML and MDA-Based Languages in Ontological Engineering

In this section we describe existing efforts to enable usage of UML, present UML tools, as well as MDA-based standards in ontological engineering. Our goal is to explain 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 UML in ontological engineering has firstly been in Cranefield's papers [13]. 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) [11]. 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, and target RDFS ontology description does not have advanced restriction concepts (e.g. multiplicity).

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	Standard UML	UML XMI	XSLT	RDFS, Java classes
Backlawski et al	UML Profile, MOF-based ontology language	(Not given - UML XMI, and MOF XMI can be used)	–	DAML
Falkovych at al	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

Backlawski and his colleagues have introduced two approaches for ontology development. The first one extends the UML metamodel by introducing new metaclasses [4]. For instance, these metaclasses 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 UML. This solution is mainly based on the DAML+OIL ontology language [41]. In order to enable usage of standard UML tools, they

propose an 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) [5]. This metamodel was also inspired by 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 and her associates [20] 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 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) 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 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 [23]. It has complex software architecture, easily extensible through plug-ins. Many components that provide interfaces to other knowledge-based tools (Jess, Algernon, OIL, Protégé Axiom Language (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 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 [22] or OWL (*http://protege.stanford.edu/plugins/owl-plugin*). 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 the Protégé (e.g. import 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'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 [21], 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. In that way, 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 (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 the Cranefield's one, since it is oriented to the 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) [44]. We believe that full usage of the recent OMG's effort – MDA [42] 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.

6. The Ontology Modeling Architecture

Currently, there is a RFP (Request for Proposal) within OMG that tries to define a suitable language for modeling Semantic Web ontology languages in the context of MDA [44]. According to this RFP, we developed ontology development architecture [17]. Of course, we do not claim that this solution is either the best one, or widest accepted one, but we only want to illustrate one of possible solutions for the OMG's initiative. One can reach other similar solutions at the OMG Ontology SIG homepage: *http://ontology.omg.org*. In our approach to ontology modeling in the scope of MDA, which is shown in Fig. 4, several specifications should be defined:

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

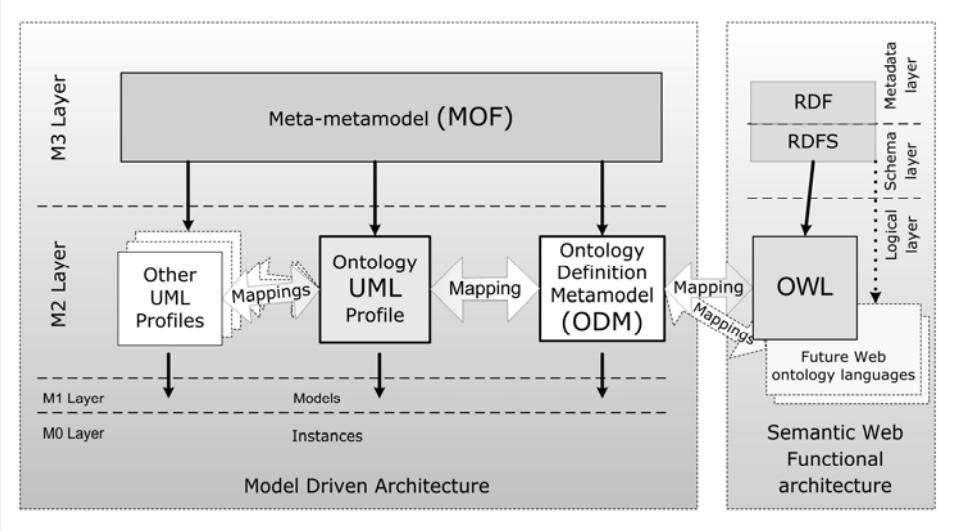

Fig. 4. Ontology modeling in the context of MDA and Semantic Web.

Ontology Definition Metamodel (ODM) should be designed to comprehend common ontology concepts. A good starting point for ODM construction is OWL since it is the result of the evolution of existing ontology representation languages, and is going to be a W3C recommendation. It is at the Logical layer of the Semantic Web [8], on top of RDF Schema (Schema layer). In order to make use of graphical modeling capabilities of UML, an ODM should have a

corresponding UML Profile [51]. This profile enables graphical editing of ontologies using UML diagrams as well as other benefits of using mature UML CASE tools. Both UML models and ODM models are serialized in XMI format so the two-way transformation between them can be done using XSLT. OWL also has representation in the XML format, so another pair of XSLTs should be provided for two-way mapping between ODM and OWL. For mapping from the Ontology UML Profile into another technology-specific UML Profiles, additional transformations can be added to support usage of ontologies in design of other domains and vice versa. We have so far implemented an XSLT that transforms the Ontology UML Profile to OWL (for details see [27]). This XSLT can be understood as an extension of present UML tools for ontology development. However, here we do not show implementation details of this transformation, but our main focus in on the MDA-compliant ontological languages.

6.1. *Metamodeling: MDA versus Functional architecture*

Before we start with more detailed description of ODM, we must clarify differences between metamodeling based on MDA, and functional architecture which is used for Web ontology languages definition. RDFS, as a schema layer language, has a non-standard and non-fixed-layer metamodeling architecture, which makes some elements in model have dual roles in the RDFS specification [47]. Therefore, it is difficult to understand by modelers, lacks clear semantics (by assigning dual roles to some elements) and propagates "layer mistake" problem to languages it defines, in our case to OWL. MDA, on the other side, has fixed and well-defined four-layer architecture. It has separate metamodeling primitives on meta-metamodel and metamodel layer that are separated from ontology language (or some other MOF-defined language) primitives, which can have infinite layers, as in the case of OWL Full.

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

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

A brief comparative description of the most important metamodeling constructs in MOF and RDF(S), which will make reading the next sections easier, is shown in Table 2. Detailed description of MOF can be found in OMG's MOF specification document [43]. RDF, RDFS and their concepts are described in detail in W3C documents [10].

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

7. Essential Ontology Definition Metamodel Concepts

This section briefly overviews the basic ODM concepts; for a more detailed description, see [17, 18]. OWL is built on top of RDFS, which is used as both modeling and metamodeling language. On the other hand, the corresponding ODM concepts are modeled by MOF. Since RDFS and MOF have numerous differences (non-fixed versus fixed metamodeling architecture [47]), OWL concepts cannot be directly copied to ODM concepts. They need some degree of adaptation.

7.1. *Resource*

OWL is built on top of RDF; thus it inherits its concepts, such as Resource, Property, metamodeling capabilities, etc. Resource is one of the basic RDF concepts; it represents all things described by RDFS and OWL. It may represent anything on the Web: A Web site, a Web page, a part of a Web page, or some other object named by URI. Compared to ontology concepts, it can be viewed as a root concept, the Thing. In RDFS, Resource is defined as an instance of `rdfs:Class`; since we use MOF as a meta-metamodeling language, Resource will be defined as an instance of MOF `Class`. It is the root class of most other basic ODM concepts that will be described: Ontology, Classifier, Property, Instance, etc. The root of this hierarchy is shown on Class Diagram in Fig. 5. Other class diagrams (shown in Figs. 6 - 8) will depict these concepts in more detail.

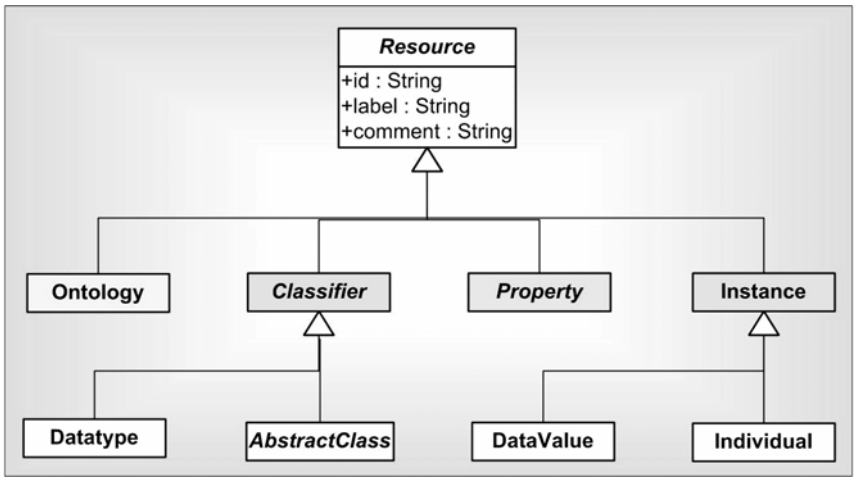

Fig. 5. The hierarchy of basic ontology concepts.

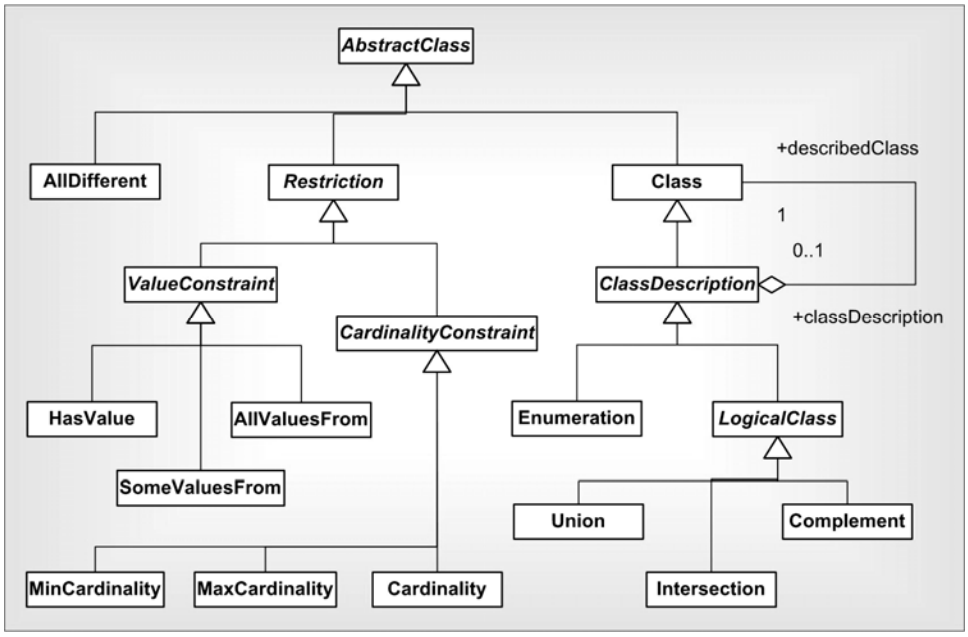

Fig. 6. The hierarchy of Ontology Classes in ODM.

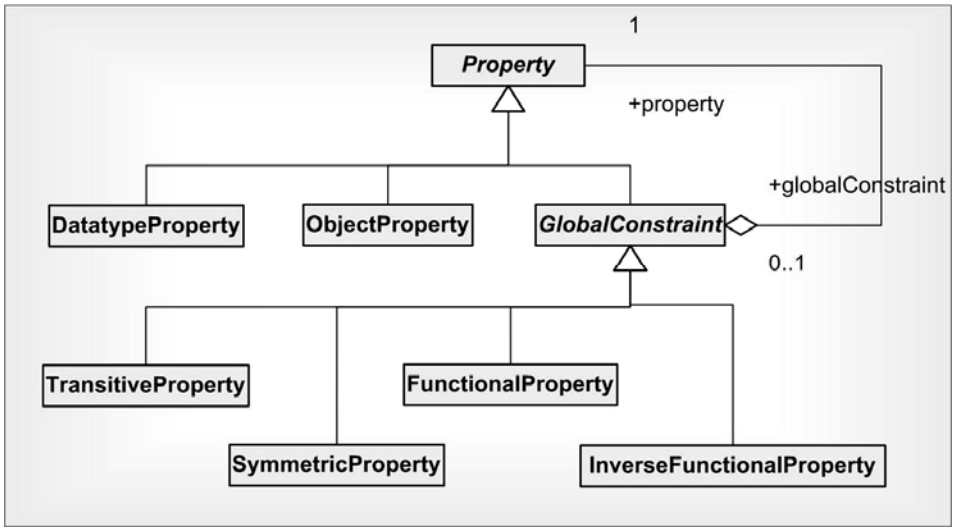

Fig. 7. The hierarchy of Ontology Properties in ODM.

`Ontology` is a concept that aggregates other concepts (`Classes`, `Properties`, etc.). It groups instances of other concepts that represent similar

or related knowledge. `Classifier` is the base class of concepts that are used for classification – `AbstractClass` and `DataType`. `Instance` is the base class of concepts that are classified by `Classifiers` – concrete `Individuals` and concrete `DataValues`. `Property` is used to represent relationships between other concepts.

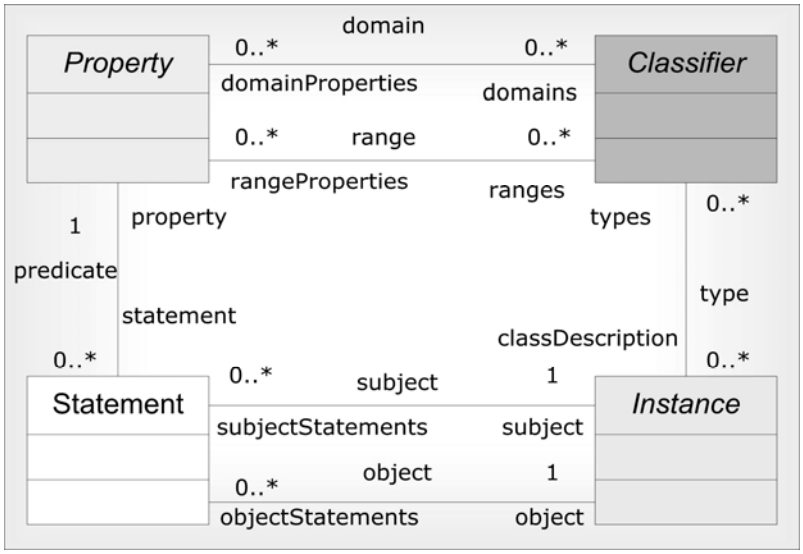

Fig. 8. Key relationships among Ontology concepts.

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

7.2. *Classifier*

In RDFS and OWL, Class (`rdfs:Class` and `owl:Class`) represents a concept for grouping resources with similar characteristics. This concept of Class (we can also call it Ontology Class) is not completely identical as a concept of Class that is defined in UML and object oriented programming languages. Every `owl:Class` is a set of individuals, called class extension. These individuals are

instances of that class. Two classes can have the same class extension but still be different classes. Ontology classes are set-theoretic, while traditional classes are more behavioral. Unlike a traditional class, an OWL class does not directly define any attributes or relations with other resources, and there is no any concept similar to methods. Attributes and relations are defined as Properties. In ODM, a Class concept corresponding to `rdfs:Class` is defined as Classifier - an instance of `MOF Class` that inherits `Resource`. A concept that complies with `owl:Class` is ODM's `AbstractClass`.

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

(i) A class can be defined by a class identifier (an URI reference) – i.e., a `Class Person`.

(ii) As an exhaustive enumeration of individuals that form the instances of a `Class`. For example, individuals `Mick, Keith, Ron, Bill` and `Charlie` form an `Enumeration – TheRollingStones.` Note that they are also members of a `Class Person`.

(iii) As a property restriction – `Class` of all individuals that have the same restriction on some of their characteristics.

(iv) As an intersection – A `Class` of all individuals that are members of all `Classes` that form an intersection. An intersection of `Classes TheWailers` and `TheRollingStones` is a `Class` that does not have any member, since no musician has played in both bands.

(v) As a union – A `Class` of all individuals that are members of any Class that forms a union. A union of `TheWailers` and `TheRollingStones`, has twelve individuals, all musicians from both bands.

(vi) As a complement – A `Class` of all individuals that are not members of other, complement class. A complement of `TheRollingStones` is a `Class` that has about six billion members – all Persons that are not members of `TheRollingStones`.

(vii) `AllDifferent` is a helper class, which states that all of its instances are have different identity.

The first concept, named class is modeled as ODM `Class`. Other five species are defined in OWL as subclasses of `owl:Class`, and are shown in Fig. 6. If we define class descriptions as simple subclasses of `Class`, like it is defined in OWL, we will have some problems related to the differences between RDFS and MOF concept of a class and the open-world assumption of the Semantic Web. While in RDFS some class instance can be easily defined to be a member of many class extensions in the same time, in MOF it can be instance of exactly one class. The open-world assumption might demand some flexibility, i.e., that class which was a `Union` becomes an `Intersection`, which is not

possible to model in MOF, since each instance can be the instance of only one `Class`, i.e., dynamic classifiers are not allowed. To solve this problem, we used the idea captured in the *Decorator* design pattern [25]. In Fig. 6, we define `ClassDescription` as a subclass of `Class` which can encapsulate a `Class`. In that way, we can have a chain of additions to the starting definition of `Class` (i.e., speaking in software engineering terms, we can add further responsibilities to the original concept of `Class`). For example, if we have some simple `Class`, we can define union by decorating that class with `Union`, and change it later to intersection, by removing the union decorator and decorating the class with `Intersection`.

7.3. *Property*

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

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

In OWL, various types of global property constraints are defined as subclasses of `Property`. Here we have the same problem we had with OWL classes, since some property might have multiple global constraints, i.e., symmetric and transitive. In this case, we also apply the Decorator design pattern, just like we did with Class Descriptions. The resulting class diagram is shown in Fig. 7. If we want to define, for example, symmetric property, we will decorate `ObjectProperty` with `SymmetricProperty`, and if we later decide that this property also should be transitive, we can simply decorate it again with `TransitiveProperty`.

7.4. *Properties predefined in RDFS and OWL*

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

Predefined properties of RDF(S) and OWL and their ODM counterparts are not completely identical. For example, the predefined property `rdf:type` states that a `rdfs:Resource` is an instance of a `rdfs:Class`. In ODM, it is represented as an `Association` between `Classifier` and `Instance`, as shown in Fig. 8, which is obviously a narrower usage than is defined in RDF. Recall that `Classifier` is further specialized in `AbstractClass` and `DataType`, and that `Instance` is specialized in `Individual` and `DataValue`. Such differences are caused by differences between MDA and Functional architecture. In RDF, `rdf:type` property is used as both metamodeling and modeling concept while in MDA, MOF is used for metamodeling, and ODM for modeling. Since ODM type association is not used for metamodeling, it is a narrower concept than `rdf:type`, thereby they are not equal.

A `Classifier` describes some general concept that has its `Instances` (`Individuals` and `DataValues`). On the other hand, a `Property` describes some generic characteristic that can describe that `Classifier` and possibly other `Classifiers`. Through `domain` we state that a Property can be used to describe a `Classifier`, and through `range` a characteristic's type. For example, a `Property nationality` can be assigned to a `Class Person` (through `domain`) with possible values which type is a `Class Country` (through `range`). In ODM, these relations are modeled as associations, as shown in Fig. 8.

7.5. *Statement*

A Statement is a Subject-Predicate-Object triple that expresses some fact in a way similar to the way facts are expressed in English. A fact that some `Individual`, `Bob` for example, has some nationality, `Jamaican`, is expressed through a Statement, which links the `Instance Bob` as the *subject*, the `nationality` property as the *predicate*, and the `Instance Jamaica` as the *object*. Thus, Statement can be viewed as some kind of `Property`'s

instance. In ODM, `Statement` is an instance of MOF `Class` that is linked with `Instance` by *subject and object* associations and with `Property` by *predicate* association (Fig. 8). ODM `Statement` slightly differs from the Statement defined in RDF (`rdf:subject` and `rdf:object` link `rdf:Statement` with `rdfs:Resource`). The difference arises from the fact that ODM is not intended for metamodeling as RDF is, similarly to the case with `rdf:type`.

8. Ontology UML Profile Essentials

In order to customize UML for modeling ontologies, we define UML Profile for ontology representation, called *Ontology UML Profile*. UML Profile is a concept used for adapting the basic UML constructs to some specific purpose. Essentially, this means introducing new kinds of modeling elements by extending the basic ones, and adding them to the modeler's tools repertoire. More details about UML extension mechanisms can be found in [35, 45]. Coherent set of extensions of the basic UML model elements, defined for specific purposes or for a specific modeling domain, constitutes a UML profile.

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

8.1. *Ontology classes*

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

In ODM, Ontology Class concept is represented as an instance of MOF Class, and has several concrete species, according to the class description: Class, Enumeration, Union, Intersection, Complement, Restriction and AllDifferent.

These constructs in the Ontology UML Profile are all inherited from the UML concept that is most similar to them, UML Class. But, we must explicitly specify that they are not the same as UML Class, which we can do using UML stereotypes. An example of Classes modeled in Ontology UML Profile is shown in Fig. 9.

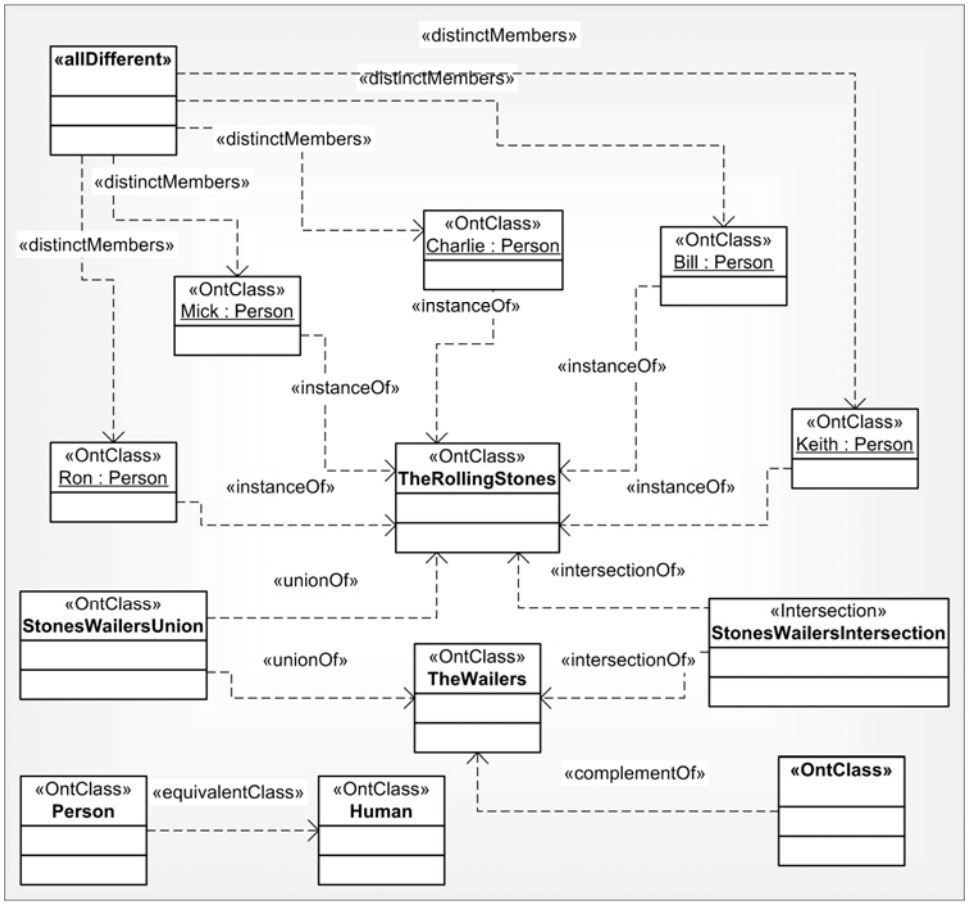

Fig. 9. Class diagram showing relations between ontology classes and individuals in the ontology UML Profile.

ODM Class identified by a class identifier will have the stereotype `«OntClass»`, `AllDifferent` - `«AllDifferent»` and Restriction - `«Restriction»`. In ODM, Enumeration, Intersection, Union and Complement are descendants of ODM Class; in Ontology UML Profile they have stereotypes

«Enumeration», «Intersection», «Union» and «Complement». The «OntClass» stereotype would be extended by each of these new stereotypes.

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

8.2. *Individuals*

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

(i) by using an underlined name of an Individual followed by ":" and its «OntClass» name (for example, Mick:Person is an Individual whose type is Person. This is the usual UML method of stating an Object's type.

(ii) by using a UML Dependency's stereotype «instanceOf» between an Individual and its «ontClass». This method is also allowed in standard UML. For example, Mick is an instance of TheRollingStones.

(iii) indirectly – through logical operators on «OntClass». If some «OntClass» is a union, intersection or complement, it is a class of Individuals that are not explicitly defined as its instances. For example, Mick is not explicitly defined as a member of StonesWailersUnion, but it is its member since he is a member of TheRollingStones, which is connected with StonesWailersUnion through a «unionOf» connection.

8.3. *Ontology properties*

`Property` is one of the most unsuitable ontology concepts to model with object-oriented languages and UML. The problem arises from the major difference between `Property` and its similar UML concepts – `Association` and `Attribute`. Since `Property` is an independent, stand-alone concept, it cannot be directly modeled with `Association` or `Attribute`, which cannot exist on their own.

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

An example of a Class Diagram that shows ontology properties modeled in UML is shown in Fig. 10. It contains four properties: Two «DatatypeProperty»s (name and socialSecurityNumber) and two «ObjectProperty»s (nationality and colleague) UML Classes. In cooperation with «domain» and «range» UML Associations, or «domain» and «range» UML Attributes, they are used to model relationships between «`OntClass`» UML `Classes`. Tagged values describe additional characteristics, for example, «`ObjectProperty`» `colleague` is symmetric (if one `Person` is a `colleague` of another `Person`, the other `Person` is also a `colleague` of the first `Person`) and transitive (if the first `Person` is a `colleague` of the second `Person`, who is a colleague of the third `Person`, the first and third `Person` are colleagues). In ODM, these characteristics are added to an ODM Class applying the Decorator Design Pattern [25]. The transformation that maps an Ontology UML Profile model to an ODM model should create one decoration of an ODM Property per attribute of Ontology UML Profile «ObjectProperty» or «DatatypeProperty».

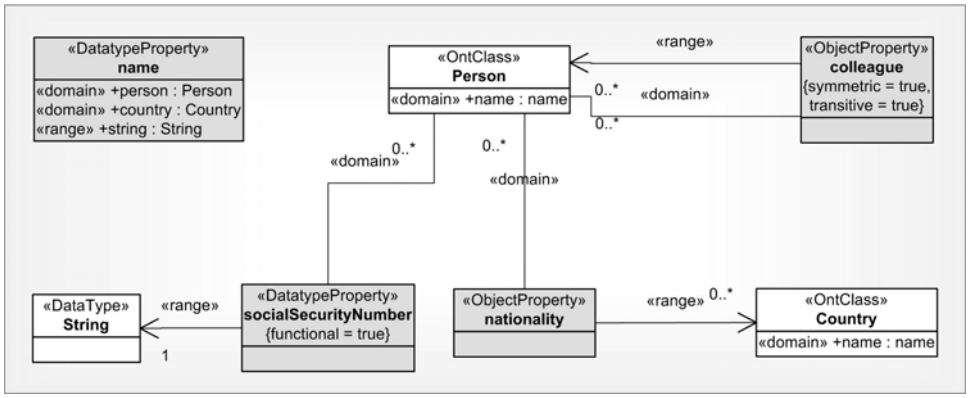

Fig. 10. Ontology properties shown in UML class diagram.

8.4. *Statement*

ODM Statement is a concept that represents concrete links between ODM instances – `Individuals` and `DataValues`. In UML, this is done through `Link` (an instance of an `Association`) or `AttributeLink` (an instance of an `Attribute`). Statement is some kind of instance of a Property, which is represented by the UML `Class'` stereotype («`ObjectProperty`» or «`DatatypeProperty`»). Since in UML a `Class'` instance is an `Object`, in Ontology UML Profile `Statement` is modeled with `Object`'s stereotype «`ObjectProperty`» or «`DatatypeProperty`» (stereotype for `Object` in UML must match the stereotype for its `Class'` stereotype). UML `Links` are used to represent the subject and the object of a Statement. To indicate that a `Link` is the subject of a Statement, `LinkEnd's` stereotype «`subject`» is used, while the object of the Statement is indicated with LinkEnd's stereotype «`object`». `LinkEnd's` stereotype is used because in UML `Link` cannot have a stereotype. These `Links` are actually instances of Property's «`domain`» and «`range`». Briefly, in Ontology UML Profile `Statement` is represented as an `Object` with two `Links` – the subject `Link` and the object `Link`, which is shown in Fig. 11. The represented `Persons` `Mick` and `Keith` are `colleagues`. They both have `UK` (Great Britain) `nationality`.

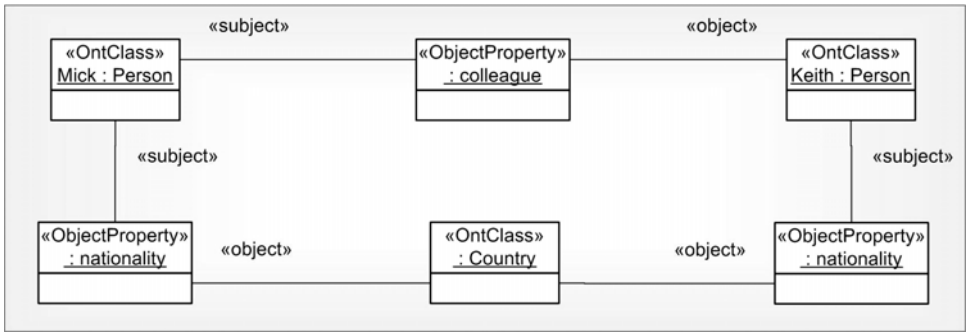

Figure 11. Individuals and Statements shown in a UML Object Diagram.

9. Conclusions

The use of software engineering techniques and standards for ontology development still requires a lot of research and work in both Semantic Web and MDA communities in order to achieve an official recommendation that will be adopted by OMG. The main task is to converge all proposed solutions that are either submitted to OMG's SIG for ontologies or published as research papers [4, 13, 17, 20]. Taking into account experience from the UML 2.0 standardization (which should be finished in 2001 [39], but it is not done yet) this can be a very long process and the date of the final recommendation is difficult to predict. On the other hand, the Semantic Web community adopted the OWL recommendation [6], and currently we have many applications that are based on ontological engineering [37].

We hope that the observation given in this chapter can be useful for the researchers from the Semantic Web community who are trying to benefit ontology development with the MDA's standards. Apart of the defined solutions for MDA-based ontology languages (Ontology Definition Metamodel and Ontology UML Profile) the practitioners need software tools that will support all these theoretical efforts. One of main tasks toward this direction is the support for transformations between Ontology UML Profile (i.e., the UML XMI format) and 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 to the OMG's ontology initiative. Until we get the formal OMG recommendation industrial engineers can use current implementations [13, 20, 27].

References

1. C. Atkinson and T. Kühne, "Rearchitecting the UML infrastructure", *ACM Transactions on Modeling and Computer Simulation* **12**(4) (2002) 290-321.
2. C. Atkinson and T. Kühne, "Profiles in a strict metamodeling framework", *Science of Computer Programming* **44**(1) (2002) 5-22.
3. C. Atkinson and T. Kühne, "Model-driven development: A metamodeling foundation"*, IEEE Software* **20**(5) (2003) 36-41.
4. K Baclawski, M. Kokar, J. E. Smith, E. Wallace, J. Letkowski, M. R. Koethe and P. Kogut, "Extending the Unified Modeling Language for ontology development", *International Journal Software and Systems Modeling (SoSyM)* **1**(2) (2002) 142-156.
5. K Baclawski, M. Kokar, J. E. Smith, E. Wallace, J. Letkowski, M. R. Koethe and P. Kogut, 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.*
6. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider and L. A. Stein, OWL Web Ontology Language Reference, *W3C Recommendation*, 2004, *http://www.w3.org/TR/2004/REC-owl-ref-20040210/.*
7. A. Bergholz, "Extending your markup: An XML tutorial", *IEEE Internet Computing* **4**(4) (2000) 74-79.
8. T. Berners-Lee, J. Hendler and O. Lassila, "The Semantic Web", *Scientific American* **284**(5) (2001) 34-43.
9. T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler, Eds., Extensible Markup Language (XML) 1.0 (Second Edition) *W3C Recommendation,* 2000, *http://www.w3.org/TR/2000/REC-xml-20001006/.*
10. D. Brickley and R. V. Guha, Eds., RDF Vocabulary Description Language 1.0: RDF schema, *W3C Recommendation,* 2004, *http://www.w3.org/TR/2000/CR-rdf-schema-20000327.*
11. B. Chandrasekaran, J. R. Josephson and V. R. Benjamins, "What are ontologies, and why do we need them?", *IEEE Intelligent Systems* **14**(1) (1999) 20-26.
12. O. Corcho, M. Fernández-López and A. Gómez-Pérez, Technical Roadmap v1.0, *OntoWeb Consortium Deliverable D11,* 2001, *http://www.ontoweb.org/download/deliverables/D11_v1_0.pdf* .
13. S. Cranefield, "Networked knowledge representation and exchange using UML and RDF", *Journal of Digital Information* **1**(8), (2001), *http://jodi.ecs.soton.ac.uk.*
14. S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Ederman and I. Horrocks, "The semantic web: The roles of XML and RDF", *IEEE Internet Computing*, **4**(5) (2000) 63-74.
15. M. Denny, "Ontology building: A survey of editing tools", 2002, *http://www.xml.com/pub/a/2002/11/06/ontologies.html.*
16. V. Devedžić, "Understanding ontological engineering", *Communications of the ACM*, **45**(4) (2002) 136-144.
17. D. Djurić, D. Gašević and V. Devedžić, "Ontology modeling and MDA", *Journal on Object Technology* **4**(1) (2005) 109-128.
18. D. Djurić, "MDA-based ontology infrastructure", *Computer Science and Information Systems* **1**(1) (2004) 91-116.
19. K. Duddy, "UML2 must enable a family of languages", *Communications of the ACM* **45**(11) (2002) 73-75.
20. K. Falkovych, M., Sabou and H. Stuckenschmidt, "UML for the semantic web: Transformation-based approaches", Eds., B. Omelayenko and M. Klein, "Knowledge

transformation for the semantic web", *Frontiers in Artificial intelligence and Applications* **95** (IOS Press, 2003) 92-106.
21. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness and P. F. Patel-Schneider, "OIL: An ontology infrastructure for the semantic web", *IEEE Intelligent Systems* **16**(2) (2001) 38-45.
22. N. Fridman-Noy, R. W. Fergerson and M. A. Musen, "The knowledge model of Protégé-2000: Combining interoperability and flexibility", *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management*, Juan-les-Pins, France (2000) 17-32.
23. N. Fridman-Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson and M. A. Musen, "Creating semantic web contents with Protégé-2000", *IEEE Intelligent Systems* **16**(2) (2001) 60-71.
24. N. Fridman-Noy and C. D. Hafner, "The State of the art in ontology design: A survey and comparative review", *AI Magazine* **18**(3) (1997) 53-74.
25. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, Reading, 1995).
26. D. Gašević, V. Damjanović and V. Devedžić, "Analysis of the MDA standards in ontological engineering", *Proceedings of the Sixth International Conference of Information Technology*, Bhubaneswar, India (2003) 193-196.
27. D. Gašević, D. Djuric, V. Devedžić and V. Damjanović, "Converting UML to OWL ontologies", *Proceedings of the 13th International WWW Conference*, New York, USA (2004).
28. A. Gómez-Pérez and O. Corcho, "Ontology languages for the semantic web", *IEEE Intelligent Systems* **17**(1) (2002) 54-60.
29. A. Gómez-Pérez (coord.), "A survey of ontology tools", *OntoWeb Consortium Deliverable 1.3,* 2002, *http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables/D13_v1-0.zip.*
30. T. R. Gruber, "A translation approach to portable ontology specifications", *Knowledge Acquisition* **5**(2) (1993) 199-220.
31. J. Hefflin and M. N. Huhns, "The zen of the web", *IEEE Internet Computing* **7**(5) (2003) 30-33.
32. J. Hendler, "Agents and the semantic web", *IEEE Intelligent Systems* **16**(2) (2001) 30-37.
33. I. Horrocks and F. van Harmelen, Eds., "Reference description of the DAML+OIL ontology markup language", 2000, *http://www.daml.org/2000/12/reference.html.*
34. I. Horrocks, "DAML+OIL: A description logic for the semantic web", *IEEE Bulletin of the Technical Committee on Data Engineering* **25**(1) (2002) 4-9.
35. J. Juerjens, *Secure Systems Development with UML* (Springer-Verlag, Berlin, 2003).
36. Y. Kalfoglou, "Exploring ontologies", Ed., S. K. Chang, *Handbook of Software Engineering and Knowledge Engineering, Vol. I – Fundamentals* (World Scientific Publishing Co., 2001) 863-887.
37. M. Klein and U. Visser, "Guest editors' introduction: Semantic web challenge 2003", *IEEE Intelligent Systems* **19**(3) (2004) 31-33.
38. M. Klein, "Tutorial: The semantic web - XML, RDF, and relatives", *IEEE Intelligent Systems* **16**(2) (2001) 26-28.
39. C. Kobryn, "UML 2001: A standardization odyssey", *Communications of the ACM* **42**(10) (1999) 29-37.
40. P. Kogut, S. Cranefield, L. Hart, M. Dutra, K. Baclawski, M. Kokar and J. Smith, "UML for ontology development", *The Knowledge Engineering Review* **17**(1) (2002) 61-64.
41. L. McGuinness, "Ontologies come of age", Eds., D. Fensel, J. Hendler, H. Lieberman and W. Wahlster, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential* (MIT Press, Boston, 2002) 171-194.

42. J. Miller and J. Mukerji, 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.*

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

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

45. OMG 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.*

46. OMG XMI Specification, v1.2*, OMG Document formal/02-01-01,* 2002, *http://www.omg.org/cgi-bin/doc?formal/2002-01-01.*

47. J. Pan and I. Horrocks, "Metamodeling architecture of web ontology languages", *Proceedings of the First Semantic Web Working Symposium*, Stanford, USA (2001) 131-149.

48. M. Ribière and P. Charlton, "Ontology overview", *Motorola Labs Paris,* 2002, *http://www.fipa.org/docs/input/f-in-00045/f-in-00045.pdf.*

49. E. Seidewitz, "What models mean", *IEEE Software* **20**(5) (2003) 26-32.

50. B. Selic, "The pragmatics of model-driven development", *IEEE Software* **20**(5) (2003) 19-25.

51. J. Siegel, "Developing in OMG's model-driven architecture", Rev. 2.6, *Object Management Group White Paper,* 2001, *ftp://ftp.omg.org/pub/docs/-omg/01-12-01.pdf.*

52. W. Swartout and A. Tate, "Guest editors' introduction: Ontologies", *IEEE Intelligent Systems* **14**(1) (1999) 18-19.