Volume 5, Issue 12, December 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Semantic Web based Model Driven Architecture

Farheen Siddiqui
Assistant Professor, Hamdard University,
New Delhi, India

Abstract: Recent efforts towards the Semantic Web vision have led to a number of standards such as OWL and Web Service languages. While these standards provide a technical infrastructure, software developers have little guidance on how to build real-world Semantic Web applications. Based on a realistic application scenario, I will be discussing thoughts on a software architecture that is built on the concept of Model Driven Architecture (MDA) and is driven by formal domain models (ontologies). We have very little excuse to build software without first doing careful design work; design not only leads to systems that are easier to develop, integrate and maintain–but also because we have the ability to automate at least some of the construction .We can take models, defined in standards like OMG's own UML, MOF and CWM, and automate the construction of data storage and application foundations. Even better, when we need to connect these "components" to each other we can automate the generation of bridges and translators based on the defining models .The promise of Model Driven Architecture to allow definition of machine readable application and data models which allow long-term flexibility of implementation, integration, maintenance and testing. The MOF-based ontology metamodels of ontology languages ODM and the Ontology UML Profile (OUP), are defined in the context of the MDA's metamodeling architecture. However, such a definition is not sufficient; they need to interact with real-word ontologies, for example with OWL ontologies. It is obvious that we need to develop transformations to support conversions between MDA ontology languages and OWL.In this paper I will describe all these transformations in terms of the modeling and technical spaces which will upgrade MDA to semantic web based MDA.

Keywords: MDA, Ontology, metamodel, semantic web

I. INTRODUCTION

The goal of the Semantic Web initiative [1] is to provide an open infrastructure for intelligent agents and Web Services. This infrastructure is based on formal domain models (ontologies) that are linked to each other on the Web. These linked ontologies provide the applications with shared terminologies and understanding. The W3C has recently finalized the Web Ontology Language (OWL) [2] as the standard format in which ontologies are represented online. Similar standardization efforts such as SWRL 1 and SCL 2 are well underway. While a lot of effort is being devoted to defining these languages and appropriate tool support, work on development methodologies for Semantic Web applications is still in its infancy. As an initial effort, W3C has recently started a working group to explore best practices and design patterns for OWL 3. However, this group focuses on ontology construction and does not help with more general issues on software architecture, use of ontologies in applications, and software testing. Most non-trivial Semantic Web applications will consist of conventional components developed in languages such as Java, and comprehensive methodologies are needed that integrate these components in a Semantic Web context. Also, while the potential benefits of ontologies in general-purpose software technology have been widely discussed [3] ontologies have not achieved a major breakthrough yet. This paper discusses state of art synergies between MDA and semantic web technology and is organized as follows: section 2 discusses about semantic web reference model , section 3 discusses the OMG model driven architecture , section 4 discusses about the concept of ontologies, section 5 related ontology to metamodel and section 6 discuss abt the mappings in MDA with ontology as model and metamodel. Finally conclusion of the paper is presented.

II. SEMANTIC WEB

In this section we want to introduce the reader to the architecture and languages of the Semantic Web . The left hand side of Figure 2 shows the static part of the Semantic Web,3 its language layers. Unicode, the URI and namespaces (NS) syntax and XML are used as a basis. XML's role is limited to that of a syntax carrier for data exchange. XML Schema [4] defines simple data types like string, date, or integer.

The Resource Description Framework (RDF) may be used to make simple assertions about Web resources or any other entity that can be named. A simple assertion is a statement that an entity has a property with a particular value, for example, that this article has a title property with value "Supporting application development in the Semantic Web." RDF Schema extends RDF by class and property hierarchies that enable the creation of simple ontologies. RDFand RDFS are already standardized by theWorldWideWeb Consortium (W3C).

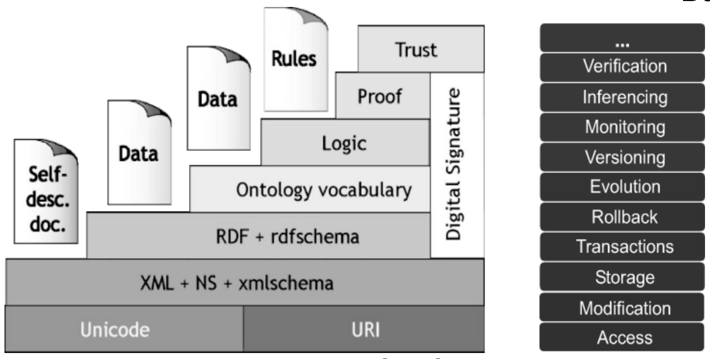

Fig 1: semantic web architecture

The Ontology layer features the Web Ontology Language.OWL is a family of richer ontology languages consisting of OWL Lite, DL and Full. They augment RDF Schema and are based on the descriptions logics (DL) paradigm [5]. OWL Lite is the simplest of these. It is a limited version of OWL DL, enabling simple and efficient implementation. OWL DL is a richer subset of OWL Full for which reasoning is known to be decidable so complete reasoners may be constructed, though they will be less efficient than an OWL Lite reasoner. OWL Full is the full ontology language, which is however undecidable. The Logic layer will provide an interoperable language for describing the sets of deductions one can make from a collection of data—how, given an ontology-based information base, one can derive new information from existing data. The Proof language will provide a way of describing the steps taken to reach a conclusion from the facts. These proofs can then be passed around and verified, providing short cuts to new facts in the system without having each node conduct the deductions themselves. The Semantic Web's vision is that once all these layers are in place, we will have an environment in which we can trust that the data we are seeing, the better description of this layer would be "Rule layer," as the Ontology layer already features a logic calculus with reasoning capabilities. We here use the naming given by Tim Berners-Lee in his roadmap.
The right hand side of Figure 1 depicts the Semantic Web's dynamic aspects that apply to data across all layers. Transactions and rollbacks of Semantic Web data operations should be possible, following the well known ACID properties (atomicity, consistency, independence, durability) of Database Management Systems (DBMS). Evolution and versioning of ontologies are important aspects, because ontologies usually are subject to change .As in all distributed environments, monitoring of data operations becomes necessary for security reasons. Finally, reasoning engines must be applied for the deduction of additional facts as well as for semantic validation.

III. MODEL DRIVEN ARCHITECTURE (MDA)

In the ideal sense, computing should be viewed by the users as "my" world, with no artificial barriers of operating system, hardware architecture, network compatibility, or application incompatibility. Given the continued, and growing, diversity of systems, this will never be achieved by forcing all software development to be based on a single operating system, programming language, instruction set architecture, application server framework or any other choice. There are simply too many platforms in existence, and too many conflicting implementation requirements, to ever agree on a single choice in any of these fields. We must agree to coexist by translation, by agreeing on models and how to translate between them. Model-driven engineering (MDE) is considered an alternative to traditional code-based software development due to its potential to increase software productivity and quality [6].
Although the architectural framework of the OMG has changed over time, the primary goals of interoperability and portability have not. The vision of integrated systems, applications that can be deployed, maintained and integrated with far less cost and overhead than that of today, is within our grasp. The Model-Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. MDA provides an approach for, and enables tools to be provided for:

- specifying a system independently of the platform that supports it,
- specifying platforms,
- choosing a particular platform for the system, and
- transforming the system specification into one for a particular platform.

The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns.A *model* of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.MDA is an approach to system development, which increases the power of models in that work. It is *model-driven* because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. The Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships of the different kinds of models.
A *viewpoint* on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system. Here 'abstraction' is used to mean the process of suppressing selected detail to establish a A viewpoint model or *view* of a system is a representation of that system from the perspective of a chosen viewpoint.

A *platform* is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.*Platform independence* is a quality, which a model may exhibit. This is the quality that the model is independent of the features of a platform of any particular type.Like most qualities, platform independence is a matter of degree. So, one model might only assume availability of features of a very general type of platform, such as remote invocation, while another model might assume the availability a particular set of tools for the CORBA platform. Likewise, one model might assume the availability of one feature of a particular type of platform, while another model might be fully committed to that type of platform.

A. MDA Viewpoints

Computation Independent Viewpoint

The *computation independent viewpoint* focuses on the on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined. A *computation independent model* is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification. It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM. The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other

Platform Independent Viewpoint

The *platform independent viewpoint* focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another. A platform independent view may use a general purpose modeling language, or a language specific to the area in which the system will be used. A *platform independent model* is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence so as to be suitable for use with a number of different platforms of similar type. A very common technique for achieving platform independence is to target a system model for a technology-neutral virtual machine. A virtual machine is defined as a set of parts and services (communications, scheduling, naming, etc.), which are defined independently of any specific platform and which are realized in platform-specific ways on different platforms. A virtual machine is a platform, and such a model is specific to that platform. But that model is platform independent with respect to the class of different platforms on which that virtual machine has been implemented. This is because such models are unaffected by the underlying platform and, hence, fully conform to the criterion of platform independence

Platform Specific Viewpoint

The *platform specific viewpoint* combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system. A *platform specific model* is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

B. Platform Model

A *platform model* provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application. *Model transformation* is the process of converting one model to another model of the same system.The drawing is intended to be suggestive. The platform independent model and other information are combined by the transformation to produce a platform specific model.The drawing is also intended to be generic. There are many ways in which such a transformation may be done. However it is done, it produces, from a platform independent model, a model specific to a particular platform.

A model is an abstraction that represents some view on reality, necessarily omitting details, and for a specific purpose—for example, [9] defines it formally as “A model is an abstraction of reality according to a certain conceptualization Kuhne [10] defines a model as “an abstraction of a (real or language based) system allowing predictions or inferences to be made”. A model may be used to document existing situations (descriptive mode) or to describe situations that have yet to eventuate (prescriptive mode) [11]. Furthermore, a model refers to a specific domain of discourse and usually adheres to the closed world assumption Karagiannis et al. [12] differentiate between iconic (non-linguistic) and linguistic models, arguing that the latter are more common in computing.

A metamodel is also a model. It is generally defined as a “model of models. In other words, it is both a model and a modelling language [13]. There is a one-to-many relationship between any element in a metamodel to corresponding elements in the model—often describing a homomorphism relationship. A model is said to conform to its metamodel when each element in the model maps to a corresponding and definitional element in the metamodel.Recent work focus on generation of metamodel from model [8]. For example, if Class exists in the metamodel, then a Bank class, a Customer class and an ATM class can all exist in the model space. It can then be said that each of these three classes (Bank, Customer and ATM) map to the Class concept in the metamodel object-oriented relationship of instantiation. However, instantiation is only relevant between a class and an object (as an instance of the class) whereas here both model elements and the corresponding metamodel elements are all classes. The importance of using conformance for model-metamodel links rather than instantiation.In metamodelling in current SE, one of two possible stacked architectures is commonly used (Figs. 2). It is worth noting that the OMG architecture in Fig. 2 is based on strict metamodelling wherein the only inter-level relationship permitted is said to be "instance of". In OMG standards, an M0 object is said to be an instance of a class in level M1; a class in level M1 is said to be an instance of a metaclass in level M2 ; and so on. However, since entities in level M2 are actually classes, the instantiation relationship read in reverse implies that M1-level entities must be objects, i.e. they are classes in relation to level M0 but objects in relation to level M2.

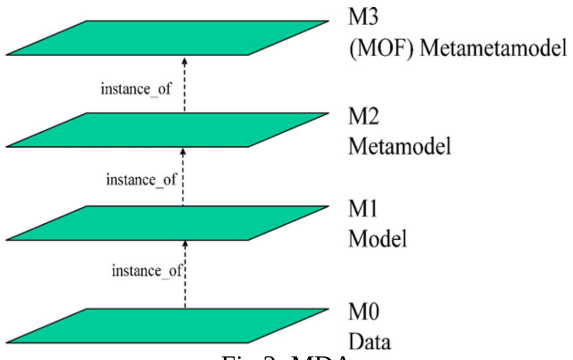

Fig 2: MDA

IV. ONTOLOGIES

In philosophy, the terms ontology and epistemology are well defined as the study of (a) existence and (b) of (human-created) knowledge, respectively. Unfortunately perhaps, other writers in software engineering (actually the majority) have taken to using the term ontology to mean epistemology (in the sense of a study of human knowledge) or, sometimes, as a noun describing the output of that aforementioned study the hierarchical or network representation of a domain of knowledge.Guizzardi [14] stress the difference between "ontology" and "an ontology", the latter referring to this aforementioned work product. An ontology can range in expressivity from a Taxonomy (knowledge with minimal hierarchy or a parent/child structure), to a Thesaurus (words and synonyms), to a Conceptual Model (with more complex knowledge), to a Logical Theory (with very rich, complex, consistent and meaningful knowledge). "Ontology" in association with "software engineering" is becoming increasingly commonplace in the literature .These ideas are then applied to the early requirements phase in Beydoun et al. [15]. When used in software engineering, additional characteristics for an ontology are generally required. From a computer science viewpoint, an ontology, to be useful, must be "understandable" by a computer—this is usually termed "formal". OMG (2005b) interprets this to mean that there must be a set of definitional rules, as specified in a metamodel: for example, by the Ontology Definition Metamodel (ODM); "formal" here really means "having form" rather than meaning precise or mathematical. A second required characteristicis that it should represent shared knowledge Finally ,it is essential that an ontology is natural language independent, it should not be equated with a vocabulary (or any other construct in a particular natural language). An ontology has the property of upholding the so-called open-world assumption, i.e. anything not explicitly expressed by an ontology is unknown. Some additional clarity is added in the various classification schemes that have been published.

V. METAMODELS AND ONTOLOGIES

To fully explore the potential use of metamodels and ontologies in software engineering, a broad understanding of the subtle overlaps between the views of the various communities working on metamodels and on ontologies is both instructive and vital. While software engineering metamodels have played a large part in standards communities such as that of the Object Management Group (OMG), much of our understanding of ontologies has been derived from the AI (artificial intelligence) community. This leads to the need to evaluate the relationship between ontologies and a metamodelling hierarchy such as that of the OMG or the International Organization for Standardization.Indeed, We seek to establish a formal relationship between metamodels and ontologies in order that the adoption and integration of ontological thinking and theory into software engineering will result in theoretically sound software development methodologies that are also practical for industry usage.

As well as comparisons of ontologies and models in the literature, several authors directly investigate comparisons of ontologies and (conceptual) metamodels—as we do here. However, manyauthors note a continuing confusion between the terms metamodel and ontology. They posit that this is because they are often depicted with the same language. They suggest differences based on focus (metamodels seek to improve similar but different models whereas an ontology does this for knowledge models) and, in their mind most importantly, on the nature of their application, arguing that an ontology is descriptive (of the problem domain) whereas ametamodel is prescriptive, belonging to the solution domain. In an example of an apparently inconsistent approach, Dillon et al.'s [16] appendices, graphically depicting their ontologies, would appear to be at the M2 level , with classes only valid at the M1 level, is therefore not in accordance with the UML language specification.

The term "ontology" can be applied at several "metalevels", as can the term "model". Of specific interest to software engineering are domain ontologies (that parallel analysis and design models) and foundational ontologies (a.k.a. meta-ontologies) that have similar characteristics to modeling languages and metamodels. It is also generally agreed that both a model and a domain ontology (M1 or Method Domain) must be represented by a language, itself defined by a metamodel

(M2). Ontologies and models are connected using a semantic mapping Metamodel-defined languages commonly used for ontology descriptions include DL (Description Logics) or RDF2/OWL (OMG, 2005b) or a proposed extension to UML provide reasoning support not possible in software modelling languages such as UML or ER (OMG, 2005b) but necessary for ontologies.

VI. MAPPINGS OF MDA-BASED LANGUAGES AND ONTOLOGIES

The MOF-based ontology metamodels of ontology languages , namely the ODM and the Ontology UML Profile (OUP), are defined in the context of the MDA's metamodeling architecture. However, such a definition is not sufficient; they need to interact with real-word ontologies, for example with OWL ontologies. It is obvious a need to develop transformations to support conversions between MDA ontology languages and OWL. In fact, this has also been requested by the OMG ODM RFP, but neither the present approach nor the current ODM draft specification itself provides a thorough solution to this problem, i.e., first proposing a conceptual solution and then digging into implementation and technological details.Accordingly, first task is to identify all modeling spaces related to this problem and depict their mutual relations. Using those relations, there are existing tools and techniques for implementing these transformations, and finally there is a describtion of one of such implementation.
Figure shows all modeling spaces that have been recognized as important for the MDA standards and the present Semantic Web ontology languages (OWL and RDF(S) in the first place) to be used cooperatively.
In the MOF modeling space, we have the ODM and the OUP. It is important to note that the ODM is defined in the M2 layer, while the OUP resides in both the M1 and M2 layers according to [Atkinson & Kühne,2002]. In Figure 3, it is shown both in the M2 layer for the sake of clarity. Concrete real-world models are located in the M1 layer and consist of classes and their instances. According to [Atkinson & Kühne, 2003], there must have more than one ontological layer in the M1 layer, and in the case of UML we have two ontological layers: one for classes and one for class instances (i.e., objects). For all MDA layers, one can use XMI, an XML compatible format for sharing metadata.In the bottommost layer (M0, not shown in the figure, but below M1), these are things from reality.

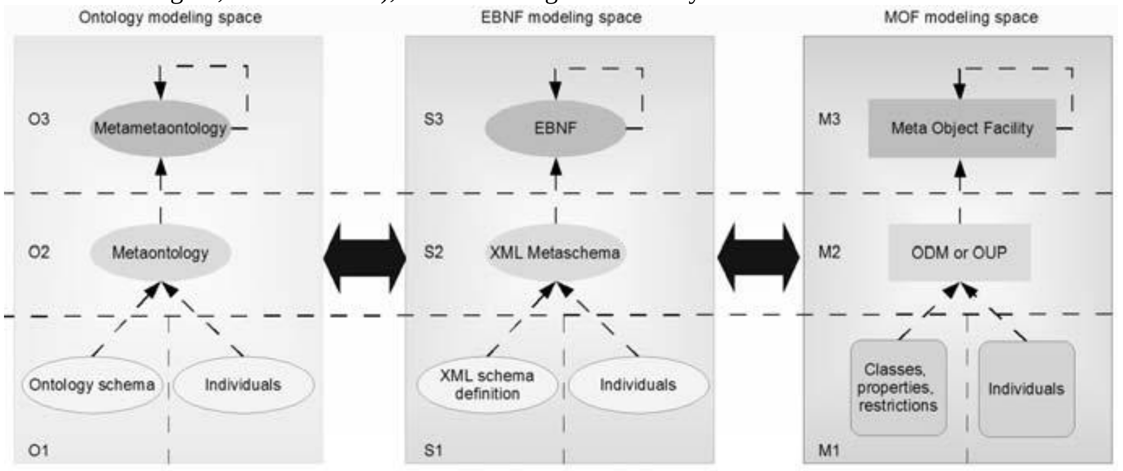

Fig. 3 M2-based mappings between the ontology modeling space and the MOF modeling space through the EBNF modeling space

The ontology modeling space includes the W3C recommendation for the Web Ontology Language (OWL). This ontology language is based on XML and RDF(S), and thus an XML format is being used for exchanging OWL ontologies. In this modeling space, we can identify various abstraction layers in order to find its relations to the MOF modeling space. The layer above bottommost is denoted as O1. In the O1 layer we build ontologies, i.e., we create classes, properties, relations, and restrictions. Ontological instances are located in the O1 layer in the ontology modeling space. We use an analogy between the topmost layer defined in the ontology modeling space. There is a metaontology that defines OWL and that this metaontology is in the O2 layer. We also refer to this modeling space, as well as the RDF(S) modeling space relaying on a similar assumption. Actually, it has generalized the treatment, in that the O3 layer is defined using RDF(S) as well. Note that such a layered organization of ontologies is already known in AI as Brachman's distinction of knowledge representation systems. However, this organization does not make a separation between ontological and linguistic relations as the schema and instances of an ontology are defined in different layers.
Using definitions of ontological and linguistic relations as well as modeling spaces, we can recognize that there is actually the same situation in the ontology modeling space – two ontological layers in the O1 linguistic layer of the ontology modeling space. In the bottommost layer of the ontology modeling space (O0), there are things from reality.
We now make some importantconclusions that one must take account of in order to provide transformations between these two modeling spaces:

1. The role of the MOF (M3) is epistemologically equivalent to the role of a metametaontology (O3).
2. The role of the O2 layer (metaontology) is equivalent to that of the metamodel layer (M2) (e.g., the ODM and the OUP), that is to say, they both specify ontology languages in terms of different modeling spaces.
3. The O1 layer has a role equivalent to that of the model layer (M1) of the MOF modeling space. In fact, this conclusion comes from Atkinson and Kühne's ontological and linguistic layers [Atkinson & Kühne, 2003],

where one linguistic layer (in this case M1) can contain many ontological layers. Accordingly, the two ontological layers residing in the O1 linguistic layer (ontology schema and ontology instances) are equivalent to the classes and objects comprising the M1 layer in the MOF modeling space.

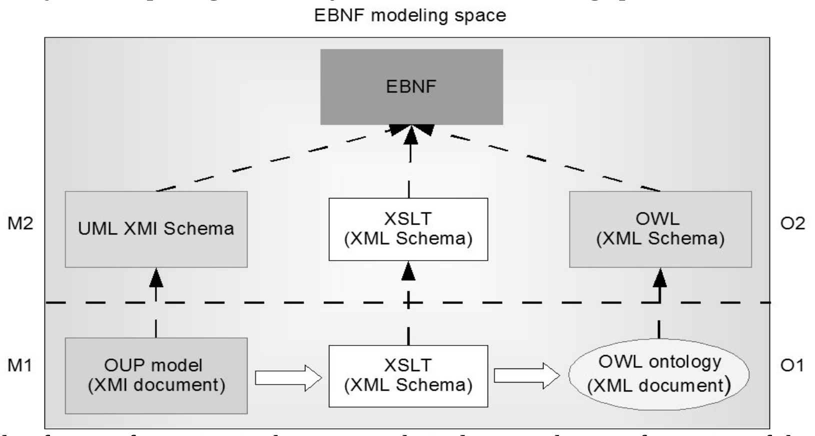

Fig 4:An example of a transformation in the XML technical space: the transformation of the OUP into OWL

Figure 4 shows an example of some transformations within the MOF modeling space. This figure is organized according to the transformation schema proposed in [Kurtev & van den Berg, 2003] for transforming XML schemas to application models. MOF modeling space can only transform those ontology languages that have an MOF-compliant metamodel. The transformation between the OUP and the ODM can be illustrated. In terms of the MOF modeling space, the transformations between these languages should be implemented in one of the QVT languages, and the chosen QVT language should have its own metamodel. Although the OUP metamodel can reside in both the M1 and M2 layer, we have placed it in the M2 layer in order to avoid problems that can arise when transforming between different metamodeling layers. As a matter of fact, this can happen if we use a standard UML profile for ontology development without the user's extensions . Note that this transformation can also be implemented through the EBNF modeling space in terms of XML schemas and XML documents.

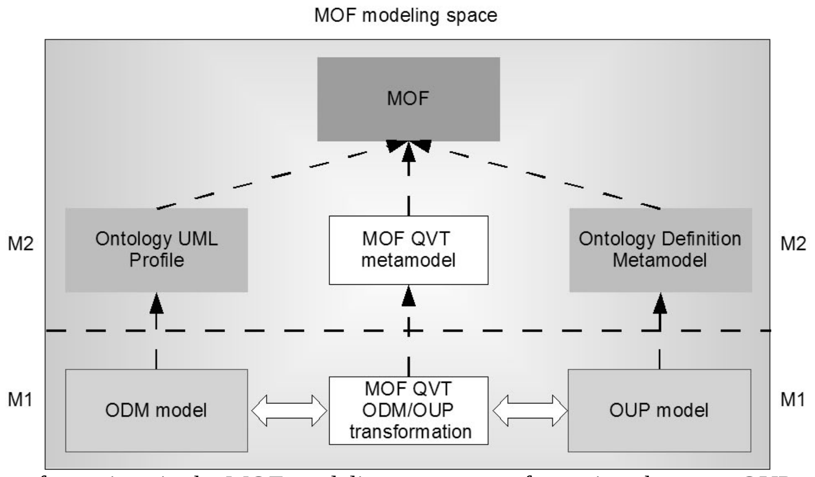

Fig 5: Transformations in the MOF modeling space: transformations between OUP and ODM

Summarizing all these facts about transformations between the ontology and MOF modeling spaces, Fig. 5 gives some guidelines on how to perform transformations between every pair of the languages discussed above. In this figure, we indicate only the transformations between the
languages considered that can be done by employing one transformation technique. At first sight, one might think that transformation between MDA-based languages (i.e., the ODM and the OUP) and OWL can be done within the MOF modeling space. However, that is not possible, since OWL is not part of the MOF modeling space, and it is not a MOF-based language. This means that a MOF-based transformation technology (e.g., QVT) cannot be applied to that pair of transformations. QVT can only be applied to MOF-based languages. So, we have to look for an intersection between those two languages. Since bridges exist between OWL and XML and between the MOF and XML, but not between OWL and the MOF, the transformation can only be defined in the EBNF modeling space as shown in Fig. 5.

VII. CONCLUSIONS

This discussion highlights the importance of high-level domain models in software development in a Semantic Web context. This approach is remarkably similar to recent developments in mainstream software technology. The Model-Driven Architecture (MDA) movement initiated by the OMG explores ways on how to better integrate high-level domain models into the development cycles of conventional software. A central idea of MDA is to employ domain models in languages like UML, and to have code generated from them for specific applications and platforms. Ontology as described above follows a very similar approach, but applies these ideas in a much more extreme way: domain models are not only used for code generation, but they are used as executable artifacts at run-time. Therefore, it is natural to assume that progress in the field of developing Semantic Web applications could be applied to MDA as well. Also, progress in MDA technology and tools may be fruitful for the Semantic Web community.

A major achievement of MDA is the grounding of the various separate modeling language standards under the umbrella of a single meta-metamodel called MOF. MOF will support tool inter-operability and standardize model translations. The OMG's recent efforts in defining a mapping between OWL and MOF/UML can become a cornerstone in bringing the fields and tools much closer together. Also, MDA advocates suggest that no single modeling formalism (such as UML class diagrams) is sufficient for domain modeling. Instead, domain-specific languages are selected depending on the task and the expertise of the domain modelers. The ontology community has made significant advances with the standardization of OWL, but OWL may not be the best modeling language for everyone. Instead, domain experts may prefer domain-specific languages or "intermediate representations" that are then translated into the details of OWL or SRWL or any other relevant language. Parts of these translations can be taken over by editing tools, by providing simplified views of lower-level modeling constructs. A common grounding of metamodels would help to formalize these views.

REFERENCES

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. Scientific American, 284(5):34–43, 2001.

[2] World Wide Web Consortium. OWL Web Ontology Language Reference. W3C Recommendation 10 Feb, 2004.

[3] Vladan Devedzi´c. Understanding ontological engineering. Communications of the ACM, 45(4):136–144, 2002.

[4] BIRON, P. V. AND MALHOTRA, A. 2001. XML Schema part 2: Datatypes. W3C Recommendation. http://www.w3.org/TR/xmlschema-2/.

[5] BAADER, F., HORROCKS, I., AND SATTLER, U. 2003. *Description Logics*. International Handbooks on Information Systems, vol. Handbook on Ontologies in Information Systems. Steffen Staab and Rudi Studer, Eds., Springer, Chapter I: Ontology Representation and Reasoning, 3–31.

[6] Schmidt, D. C.: Guest Editor's Introduction - Model-Driven Engineering. IEEE Computer, vol. 39, no. 2, Feb. 2006, pp. 25-31.

[7] Karsai, G., Neema, S., Sharp, D.: Model-driven architecture for embedded software - A synopsis and an example. Science of Computer Programming, vol. 73, no. 1, Sep. 2008, pp.26-38.

[8] Liu, Q., Bryant, B.R., Mernik, M.: Metamodel recovery from multi-tiered domains using extended MARS. In Proceedings of the 34th Annual International Computer Software and Applications Conference, Seoul, South Korea, Jul. 2010, pp. 279-288.

[9] Guizzardi, G., Wagner, G., 2005a. On the ontological foundations of agent concepts. In: Bresciani, P., Giorgini, P., Henderson-Sellers, B., Low, G., Winikoff, M. (Eds.), Agent-Oriented Information Systems II, LNCS 3508. Springer-Verlag, Berlin, pp. 113–128.

[10] Kuhne, T., 2006a. Matters of (meta-)modeling. Software and Systems Modelling , 369–385.

[11] Henderson-Sellers, B., 2007. On the challenges of correctly using metamodels in method engineering. In: Fujita, H., Pisanelli, D. (Eds.), Keynote paper in New Trends in Software Methodologies, Tools and Techniques. Proceedings of the sixth SoMeT 07. IOS Press, pp. 3–35.

[12] Karagiannis, D., Fill, H.-G., Hofferer, P., Nemetz, M., 2008. Metamodelling: some application areas in information systems. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (Eds.), UNISCON 2008, LNBIP 5. Springer-Verlag, Berlin.

[13] Bertoa, M.F. and Vallecillo, A., 2010. Quality attributes for software metamodels, Proceedings of the Workshop on Quantitative Approaches to Object-Oriented Software Engineering (QAOOSE 2010), Malaga, Spain, 2 July 2010.

[14] Guizzardi, R.S.S., Guizzardi, G., Perini, A., Mylopoulos, J., 2007. Towards an ontological account of agent-oriented goals. In: Software Engineering for Multi-Agent Systems V, LNCS 4408. Springer-Verlag, Berlin, pp. 148–164

[15] Beydoun, G., Krishna, A.K., Ghose, A., Low, G.C., 2008. Towards ontology-based MAS methodologies: ontology based early requirements. In: Barry, C., Lang, M., Wojtkowski, W., Wojtkowski, G., Wrycza, S., Zupancic, J. (Eds.), The Inter-Networked World: ISD Theory, Practice, and Education. Springer-Verlag, New York.

[16] Dillon, T.S., Chang, E., Wongthongtham, P., 2008. Ontology-based software engineering – software engineering 2.0. In: Procs. 19th Australian Software Engineering Conference ASWEC. 2008 Perth, Australia.