Ontological Perspective in Metamodeling for Model Transformations

Arda Goknil
Ege University
Computer Engineering Department
35100, Bornova, Izmir, Turkey
arda.goknil@ege.edu.tr

Yasemin Topaloglu
Ege University
Computer Engineering Department
35100, Bornova, Izmir, Turkey
yasemin.topaloglu@ege.edu.tr

ABSTRACT

Model Driven Engineering (MDE) aims to facilitate building larger and more complex, reliable software systems by introducing a higher abstraction level than the code level. The technical space concept discusses how the basic MDE principles may be mapped onto modern platform support and several technical spaces are proposed to support MDE. In this paper, we propose to use the ontology technical space in model transformations to achieve the targets of MDE. Using the ontology technical space will enable us to model not only the meta concepts but also the semantic context which can be used in model inferencing. Within this context, we define meta models of object oriented models ontologicaly.

Categories and Subject Descriptors

D.2.10 [**Design**]: representation; H.1.1 [**Systems and Information Theory**]: general systems theory

General Terms

Design, Languages, Theory

Keywords

Model Transformations, Transformation Languages, Metamodeling, Ontology

1. INTRODUCTION

Software engineering principles guided developers in building complex software systems for years. The need for larger, more complex, and also reliable software causes software development activities to get complicated. *Model Driven Engineering (MDE)* tackles this problem of software development by using models at different levels of abstraction for developing systems. Thus, the main activity of MDE developers is to design models like they used to develop code. The main idea behind this is to enable software developers to work in a higher abstraction layer than the code level. The transitions between the models in different abstraction levels and the code generation from these models are provided by model transformations. As a consequence, models and model transformations become the primary artifacts of software development [12].

The current solutions for Model Driven Engineering can be grouped in the technical spaces [13]. A technical space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [2]. The *Model Driven Architecture (MDA)* [16] approach is a technical space which was offered by the *Object Management Group (OMG)* for model driven engineering. Although several other technical spaces like the *EMF technical space*, the *Microsoft DSL Tools technical space* use different standards, they have similar goals for model driven engineering. We propose bridging between the MDA technical space and the ontology technical space in order to model not only the meta concepts but also the semantic context. In this paper, we define an ontological modeling infrastructure in the ontology technical space where all the reasoning facilities are already available. Our approach uses *Web Ontology Language (OWL)* in the ontology technical space.

An ontology is defined as a formal explicit description of concepts in a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions of slots [14]. Web Ontology Language (OWL) is a technology for ontology development and knowledge representation in Semantic Web [4]. OWL defines and instantiates Web Ontologies. Recent works [1] [7] [8] discuss that UML [15] could be a key technology for the ontology development bottleneck. A number of partial solutions are currently available as a result of these works and the OMG initialized a working group to create Ontology Definition Metamodel (ODM) to define M2 level UML-ontology-OWL transformation [18]. Alternatively to the established views, we proposed another approach for the collaboration between MDA and OWL in our previous work [9]. While recent works discuss the contributions of MDA to ontology development, we discuss the possible contributions of ontologies to MDA. We proposed an ontology based model transformation infrastructure to transform application models by using query statements, transformation rules and models defined as ontologies in OWL [9]. When we discuss the modeling standards in the technical space concept, we realize that defining transformation components in OWL is not enough for the ontology technical space.

In this paper, we discuss our ontological modeling approach in the ontology technical space and present an ontol-

Metainformatics Symposium, November 9–11, 2005, Esbjerg, Denmark.

ogy for object-oriented models. The paper is organized as follows. In Section 2, we discuss the general characteristics and underlying concepts of the ontology technical space approach. In Section 3, we present an ontological metamodel for object-oriented models. Section 4 includes the conclusions.

2. OVERVIEW OF THE APPROACH IN THE ONTOLOGY TECHNICAL SPACE

2.1 Web Ontology Language (OWL)

Web Ontology Language (OWL) is a result of the ongoing process of defining a standard ontology web language. It is an extension of *Resource Description Framework (RDF)* [22].

A Class identifier describes a named class in OWL ontology. For instance, "<owl:Class rdf:ID= "Student">" defines a class "Student" which is an instance of "owl:Class". In the ontology, many individuals can be instantiated from the defined classes.

It is possible to constrain the range of a property in specific contexts in a variety of ways [4]. The various forms can only be used within the context of a property restriction (*owl:Restriction*). Property restrictions include value (*owl:allValuesFrom*, *owl:someValuesFrom*, *owl:hasValue*) and cardinality constraints (*owl:cardinality*, *owl:maxCardinality*, *owl:minCardinality*). For example, the following description is a value constraint:

```
<owl:Restriction>
    <owl:onProperty rdf:resource="#manages"/>
    <owl:allValuesFrom rdf:resource="#Student"/>
</owl:Restriction>
```

It defines an anonymous class of all individuals whose *manages* property only has range values of class *Student*. Similarly, the following description is a cardinality constraint which defines a class of all individuals that manages one student at least:

```
<owl:Restriction>
   <owl:onProperty rdf:resource="#manages"/>
   <owl:cardinality rdf:datatype="
&xsd;nonNegativeInteger">1
   </owl:cardinality>
</owl:Restriction>
```

OWL has three class axioms for additional conditions: *owl:disjointWith*, *owl:equivalentClass*, and *rdfs:subClass*. The *rdfs:subClassOf* construct relates a more specific class to a more general class. It is the fundamental taxonomic constructor [4].

```
<owl:Class rdf:ID="University">
   <rdfs:subClassOf rdf:resource=
 "#GraduateSchool"/>
   <owl:disjointWith rdf:resource="#Institute">
</owl:Class>
```

The statement in the above defines that the class *University* is a subclass of class *GraduateSchool*, and it is disjoint with class *Institute*. There are two kinds of properties defined in OWL: *object property* which relates individuals to individuals, and *datatype property* which relates individuals to data values.

```
<owl:ObjectProperty rdf:about="#hasStudent">
   <rdfs:domain rdf:resource="#University"/>
   <rdfs:range rdf:resource="#Student"/>
</owl:ObjectProperty>
```

There are also constructors used to relate two properties (*owl:inverseOf* and *owl:equivalentProperty*), to specify cardinality constraints on properties (*owl:FunctionalProperty* and *owl:InverseFunctionalProperty*), to define logical characteristics of properties (*owl:TransitiveProperty* and *owl:SymmetricProperty*), and to relate individuals (*owl:sameAs*, *owl:sameIndividualAs*, *owl:differentFrom* and *owl:AllDifferent*).

2.2 Basic Components of the Approach

Model transformation is the core activity in MDE to generate new models or to change the existing models. A model transformation takes one or more source models as input and produces one or more models as output according to a set of transformation rules. The metamodeling technique is used to define these models and transformation rules [20]. A metamodel describes models by defining the meta entities and the relationships among these entities together with the semantics of these relationships. The meta class instances of the metamodel define the models and transformation rules generated from the metamodel. Extensible languages like *XML Metadata Interchange (XMI)* and *Extensible Stylesheet Language Transformations (XSLT)* can be used to encode models and transformation rules with meta class instances [5][21] [23].

There are various standards and tools defined for the realization of MDE principles. For example, the OMG MDA technical space is defined around the standards like MOF, XMI, OCL, UML, and CWM [16]. It has been found very convenient to use XML as a support notation for model serialization, since XML documents correspond to trees which are easier to serialize than a graph. Since XMI is a standard model serialization procedure which is based on XML for MDA models (i.e. MOF compliant-models), using XMI to serialize models provides only model representation.

In our ontology technical space, the components and standards are realized around the ontology engineering and web ontology language (OWL). We use OWL not only as a notation for model serialization. OWL vocabulary constitutes the main library for deriving models instead of MOF. It allows both representation of models and reasoning on models.

Our proposal includes ontology based definitions for the three components of a model transformation defined in MOF 2.0 Query/Views/Transformations RFP [17]. The QVT RFP is issued by the Object Management Group (OMG) and seeks a standard solution for model manipulation in the context of MDA. There are available submissions [6] to the QVT RFP. The three main areas of model transformation defined by QVT [17] are:

- *Queries*, which take a model as input, and selects specific elements from that model.
- *Views*, which are models that are derived from other models.
- *Transformations*, which take a model as input and update it or create a new model.

Although our definition does not use the standards like XMI, and MOF which are defined in the OMG MDA technical space, we describe our ontological components according

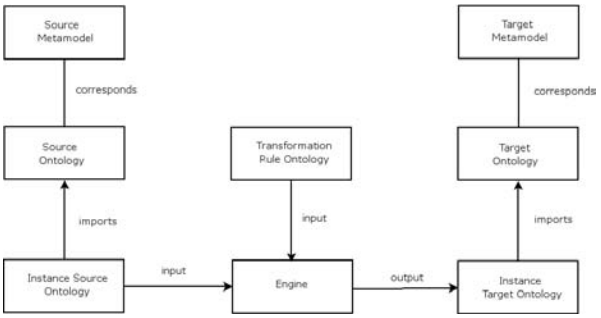

Figure 1: Overall Approach of Ontology Based Model Transformation

to the QVT parts. This provides another bridge between the OMG MDA technical space and the ontology technical space.

Figure 1 shows the overall architecture of ontology based model transformation. Source and target ontologies in our approach correspond to the source and target meta models in QVT. These ontology documents include the classes which correspond to the meta classes in the meta models. They define the main entities and the possible associations which the instance source and target ontologies use to derive the individuals. The engine transforms the instance source ontology to the instance target ontology according to the transformation rule ontology.

2.3 Related Work

There are two dimensions in the research projects about the collaboration of ontology concepts and model driven engineering. The mappings and transformations between the UML constructs and the OWL elements to develop ontologies are the main concerns of the research projects in the first dimension. Recent works [1] [7] [8] discuss that UML could be a key technology for the ontology development bottleneck. A number of partial solutions are currently available as a result of these works and Object Modeling Group (OMG) initialized a working group to create *Ontology Definition Metamodel (ODM)* to define M2 level UML-ontology-OWL transformation. ODM aims to generate ontology descriptions from UML models. On the other hand, we propose another approach for the collaboration between model driven engineering and OWL which is in the second dimension research of the ontology and model driven engineering collaboration. While recent works discuss the contributions of model driven engineering to ontology development, we discuss the possible contributions of ontologies to model driven engineering in the context of ontology based model transformations.

Roser and Bauer [19] propose a structure for ontology based model transformations. Their proposal includes a connection between syntax defined in metamodels and the semantics of the ontology elements. This approach defines a generator that takes meta models, ontologies and specifies semantic transformation to generate an intermediate model transformation language that aims to obtain a common representation of model transformations independent to specific transformation languages [19]. Their architecture includes semantic transformation and syntax-semantic binding (metamodel-ontology binding, ontology-metamodel binding). The tasks performed can be grouped as; deriving semantic information from metamodels with metamodel-ontology binding, transforming the derived ontology to target ontology with semantic transformation, and expressing ontology elements of the transformed ontology in metamodels with ontology-metamodel binding.

In our previous work [9], we proposed an ontology based model transformation infrastructure to transform application models by using query statements, transformation rules and models defined as ontologies in OWL. In [9], we defined a simple query and a transformation language as ontologically. There are two related ontology documents to query application models. The *Query Ontology* defines the main query entities and the possible associations of these entities. The *Instance Query Ontology* selects the specific elements in the application document, and it is derived from the meta entities which are defined in the *Query Ontology*. The relationship between the *Query Ontology* and the *Instance Query Ontology* is similar with the relationship between the *Source Ontology* and the *Instance Source Ontology*. When our approach is considered from the perspective of technical space concepts, also the semantic context of the models can be derived and defined as ontologies.

3. MODELING THE COMPONENTS AS ONTOLOGIES

3.1 Ontological Definition of Software Models

In Model Driven Engineering, the basic principle is that *everything is a model* [3]. Although there are different technical spaces in meta modeling for MDE, the 3+1 meta modeling hierarchy is the main concept to define models and meta models as shown in Figure 2 [2].

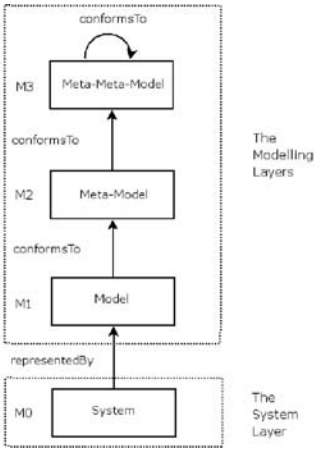

Figure 2: The Four Layer Classical Architecture [3].

The four-layer architecture is divided into two groups called the modeling layers and the system layers [2]. M0 layer is the ontological layer and represents the real world system. M1 layer represents this layer and this instantiation relationship is not the same with the instantiation relationship with

the upper layers. In this hierarchy, the meta layers do not include the domain and semantic context. It is not possible to inference on the models. This meta modeling hierarchy operates at the syntactical level [19].

In [2], it is argued that some functionalities are easier to provide in a technical space with a M3 based on OWL than on the MOF. Bezivin [2] states that, since an object may be referred by different names, OWL has a better name management. OWL also allows inferring from the properties of an individual that is a member of a class.

In our approach, models are defined as ontologies. Figure 3 shows our ontology structure to define models as ontologies and the relationship between the two technical spaces. While MOF is the base meta-meta model in the MDA technical space, OWL vocabulary constitutes the main library for derived ontologies to define models. The *Model Ontology* defines the main entities and the possible associations of these entities in the model. The main difference between the *Model Ontology* and *UML meta-model* is that *Model Ontology* does not operate at syntactical level and it includes constraints and semantic context. Unlike XMI, OWL is more than a notation for model serialization. It allows both representing models and reasoning on models. OWL vocabulary serves richer vocabulary to *Model Ontology* rather than MOF serves to M2 level. The restriction mechanism in OWL allows defining constraints about the individual ontologies derived from model ontologies and is compared with *Object Constraint Language* [24].

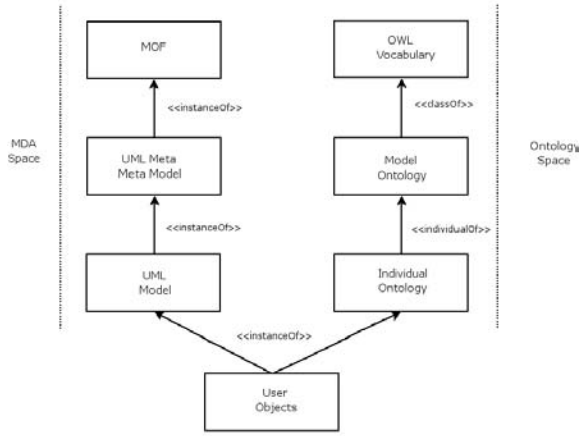

Figure 3: The Proposed Ontological Layers for Mapping MDA and Ontology Technical Spaces

We aim to define the relationship M2/M1 layers in UML and the equivalent modeling ontologies in OWL. In the Ontolology Definition Metamodel [18], it is proposed to constitute the relationship between an M1/M0 model in UML and the equivalent model in OWL, so this study on ODM will enable UML based ontology development. ODM's design rationale is mainly for ontology engineering. The main difference between ODM and our approach is that we mainly study on M2 layer while ODM is working on M1 layer. ODM aims to support generation of ontology descriptions from UML models. In our approach, we try to define UML constructs in OWL. This approach enables us to study in the M2 layer instead of M1 layer.

3.2 Representation of an Object Oriented Metamodel Ontologically

3.2.1 Model Ontology Definition

The model ontology defines the main entities and the possible associations of these entities in the model. Figure 4 shows a simplified object oriented metamodel which we define ontologically in our *Model Ontology*.

We define the basic modeling elements in UML ontologically in the *Model Ontology*. Both UML and OWL are based on classes. An OWL class is declared by assigning a name to the relevant type. Every class in Figure 4 is assigned to the owl:Class by giving names in our *Model Ontology*. For example

```
<owl:Class rdf:ID="Class"/>
<owl:Class rdf:ID="Method"/>
```

The *owl:Class* comes from the OWL vocabulary and it is used to define the types of the *Class* and *Method* classes in the *Model Ontology*. This derivation is similar to *instance of* relation between M3 and M2 layers.

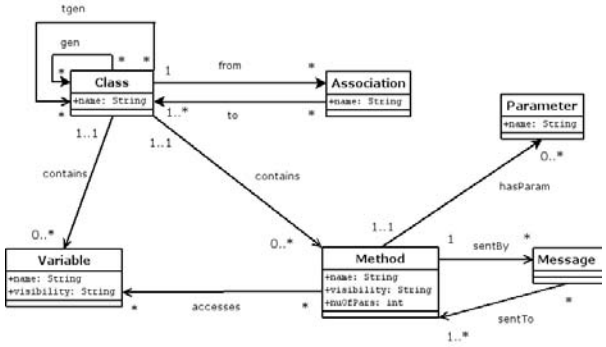

Figure 4: Example of A Simplified Object Oriented Metamodel.

Relationships among classes in OWL are called properties. A binary association translates directly to an *owl:ObjectProperty*. If the association name occurs more than once in the same model, it must be disambiguated in the OWL [18]. In Figure 4, the *Contains* relationship occurs more than once in the model. In our *Model Ontology*, we could not use the same name "contains" in more than one object property so that we concatenate the member names to the association name.

```
<owl:ObjectProperty rdf:about="#containsVar">
    <rdfs:domain rdf:resource="#Class"/>
    <owl:inverseOf rdf:resource=
  "#containedByCls"/>
    <rdfs:range rdf:resource="#Variable"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="containsMet">
    <rdfs:domain rdf:resource="#Class"/>
    <rdfs:range rdf:resource="#Method"/>
    <owl:inverseOf rdf:resource=
  "#containedByClass"/>
</owl:ObjectProperty>
```

The owl property always has a domain and range specified. The *containsVar* object property has a domain named *Class* and a range named *Variable*. The restriction mechanism in OWL can be used to define the cardinality constraint in the relation. *owl:cardinality*, *owl:maxCardinality* and *owl:minCardinality* permit the specification of the number of elements in a relation. The relation between *Class* and *Variable* is a one-to-many relation. The cardinality constraint in *containsVar* object property defines the multiplicity of *Variable* because the range in this property is *Variable*. The multiplicity of the *Variable* class is many so that we do not have any cardinality constraint in *containsVar* property. But one object property is not enough to define a relationship in M2 level. For example we can not restrict the cardinalities of both nodes in one object property. For our case we have another object property which is the inverse of *containsVar* property.

```
<owl:ObjectProperty rdf:about=
  "#containedByCls">
     <rdfs:domain rdf:resource="#Variable"/>
     <rdfs:range rdf:resource="#Class"/>
     <owl:inverseOf>
       <owl:ObjectProperty rdf:ID=
  "containsVar"/>
     </owl:inverseOf>
</owl:ObjectProperty>
```

containsByCls object property has a domain named *Variable* and a range named *Class* and it is the inverse of *containsVar* property. The *owl:inverseOf* construct is used to define such an inverse relation between two properties and these two properties constitute the contains relationship between the *Class* and *Variable* classes. The cardinality constraint of *Class* class in this relation is defined as *owl:Restriction* in *containedByCls* object property.

```
<owl:Restriction>
    <owl:cardinality rdf:datatype=
  "http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID=
            "containedByCls"/>
        </owl:onProperty>
</owl:Restriction>
```

The attributes whose types are primitive data type in the *Model Ontology* are defined as *owl:DatatypeProperty*.

```
<owl:DatatypeProperty rdf:ID="nuOfPars">
    <rdfs:range rdf:resource=
      "http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#Method"/>
</owl:DatatypeProperty>
```

3.2.2 Individual Ontology Definition

Individual Ontology corresponds to user defined object oriented models. Figure 5 shows a basic object oriented model which defines *Student-Book* classes and the relation between them in UML.

The base model elements are in the *Model Ontology* and instance models are defined in the *Individual Ontology* by deriving the base classes from the *Model Ontology*. The *individualOf* relation in Figure 3 specifies this derivation process. It is possible for ontologies to be treated as reusable

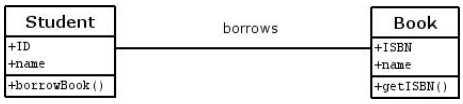

Figure 5: A Simple Object Oriented Model

modules and imported into different documents. An OWL document may contain an individual of class defined in another ontology, which contains meta-data about that document itself. In our case, the *Individual Ontology* defining the *Student-Book Model* imports the *Model Ontology* to create individuals as shown below:

```
<owl:Ontology rdf:about="">
    <owl:imports rdf:resource="Model.rdf"/>
</owl:Ontology>
```

The *owl:imports* construct allows to include by reference in a knowledge base the axioms contained in another ontology. Using an *owl:imports* statement is the fact that with "*imports*" both ontologies stay in different files. There is also another ongoing work to provide modelers with suitable means for developing ontologies in a modular way and to provide an alternative to the *owl:imports construct* [10]. This ongoing project [10] will provide us both syntactic and logical modularity in combining the *Model Ontology* and the *Individual Ontology*.

The *Individual Ontology* contains the instances which belong to the user model. To represent the object oriented model in Figure 5, we use the base classes in the *Model Ontology* defined in Figure 4.

```
<Class rdf:ID="Book">
   <name rdf:datatype=
     "http://www.w3.org/2001/XMLSchema#string">
       Book</name>
```

In Figure 5, the *Book* class has two variables and one method. These variables and method individuals are linked with the *Book* class with appropriate object properties derived from the *Model Ontology*.

```
<containsVar>
      <Variable rdf:ID="BookName">
        <Vvisibility rdf:datatype=
     "http://www.w3.org/2001/XMLSchema#string"
        >True</Vvisibility>
        <Vname rdf:datatype=
     "http://www.w3.org/2001/XMLSchema#string"
        >name</Vname>
        <containedByCls rdf:resource="#Book"/>
      </Variable>
</containsVar>

<containsMet>
      <Method rdf:ID="getISBN">
        <Mname rdf:datatype=
     "http://www.w3.org/2001/XMLSchema#string"
        >getISBN</Mname>
        <Mvisibility rdf:datatype=
     "http://www.w3.org/2001/XMLSchema#string"
        >True</Mvisibility>
        <containedByClass rdf:resource="#Book"/>
      </Method>
</containsMet>
```

The *containsVar* tag is inside the *Book* class and this tag contains the declaration of the *BookName* variable individual. In the *Individual Ontology*, there is no need to declare

the inverse object property of *containsVar* or *containsMet* object properties. Their inverse object properties are used to restrict the cardinality of the domains of these object properties. To constitute the relation between the *Student* and *Book* classes, we create an individual named *borrows* from the *Association* base class. This individual has two object property named *from* and *to* with their inverse object properties. We omit the *AssociationEnd* class in the *Model Ontology* to simplify the *Model Ontology*. The hierarchy between the classes in the ontology as shown below:

```
<Association rdf:ID="borrows">
      <to>
        <Class rdf:ID="Book">
          <invTo rdf:resource="#borrows"/>
      </Class>
      </to>
    <invFrom>
        <Class rdf:ID="Student">
          <from rdf:resource="#borrows"/>
        </Class>
    </invFrom>
</Association>
```

Some functionalities are easier to provide with this M3 based on OWL approach than on the MOF. OWL itself has a restriction mechanism compared with other predicate definition languages like OCL. Modeling and restricting shared knowledge in OWL allows us to define more specialized domains in *platform independent models*. In the *Model Ontology*, we can define a domain class as the set of individuals which satisfy a restriction expression. These expressions can be a boolean combination of other classes (*intersectionOf*, *unionOf*, *complementOf*), or property value restriction on properties [18]. OWL allows us to define these kinds of restrictions on public and shared domain classes.

4. CONCLUSION

In this paper, we presented a semantic perspective in meta modeling for model transformations based on ontologies. We provided ontological model definitions that can be used in model transformations. We used OWL in the definition of ontologies since OWL is executable and also supported by various tools. Using OWL in model transformation infrastructure allows us to use the current semantic technologies for constituting the transformation engines. Some programmatic environments [11] include OWL APIs which provide persistent storage, reading and writing OWL documents. Loading and compiling the parts of model transformation can be processed by the help of current ontology APIs.

It is our belief that that ontology technical space will play an important role in the development of model driven engineering. The ontological metamodeling infrastructure we propose will enable to inference on various models. In our future work, we will investigate possible inference opportunities on ontology-based models and other ontology-based transformation alternatives. Ontology technologies like ontology mapping and inference engines will support us in model inferencing, management and model transformations. Now defining models as ontologically gives us a great opportunity about inferencing on the software models.

5. REFERENCES

[1] Kenneth Baclawski, Mieczyslaw M. Kokar, Paul A. Kogut, Lewis Hart, Jeffrey E. Smith, Jerzy Letkowski, and Pat Emery. Extending the unified modeling language for ontology development. *Journal on Software and System Modeling*, Vol.1(2):142–156, 2002.

[2] Jean Bezivin. Model driven engineering: Principles, scope, deployment and applicability. Technical Report TR-01-01, Summer School on Generative and Transformational Techniques in Software Engineering, 2005.

[3] Jean Bezivin. On the unification power of models. *Software and System Modeling*, Vol.4:171–188, 2005.

[4] Mike Dean, Guus Schreiber, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. *OWL Web Ontology Language Reference W3C Recommendation*. W3C, February 2004.

[5] Birgit Demuth, Sven Obermaier, and Heinrich Hussmann. Experiements with xmi based transformations of software models. In *Proeedings of WTUML: Workshop on Transformations in UML*, Genova, Italy, April 2001.

[6] Keith Duddy, Anna Gerber, Michael Lawley, Kerry Raymond, and Jim Steel. Model transformation: A declarative, reusable patterns approach. In *EDOC*, pages 174–185. IEEE Computer Society, 2003.

[7] Kateryna Falkovych, Marta Sabou, and Heiner Stuckenschmidt. Uml for the semantic web: Transformation-based approaches. In *Knowledge Transformation for the Semantic Web*, pages 92–106. 2003.

[8] Dragan Gasevic, Dragan Djuric, Vladan Devedzic, and Violeta Damjanovic. Approaching owl to mda through technological spaces. In *Essentials of the 3rd UML Workshop in Software Model Engineering (WISME 2004)*, 2004.

[9] Arda Goknil and N. Yasemin Topaloglu. Ontology based model transformation infrastructure. In Savitri Bevinakoppa, Luís Ferreira Pires, and Slimane Hammoudi, editors, *WSMDEIS*, pages 55–64. INSTICC Press, 2005.

[10] Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using e-connections. *Journal of Web Semantics*, Vol.4:42, 2005.

[11] HP-Labs. Jena - a semantic web framework for java, available at http://jena.sourceforge.net/.

[12] Sheena R. Judson, Robert B. France, and Doris L. Carver. Specifying model transformations at the meta-model level. In *Essentials of the 2nd UML Workshop in Software Model Engineering (WISME 2003)*, 2003.

[13] Ivan Kurtev, Jean Bezivin, and Mehmet Aksit. Technical spaces: An initial appraisal. In *Tenth International Conference on Cooperative Information Systems (CoopIS), Federated Conferences Industrial Track*, California, 2002.

[14] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory, March 2001.

[15] OMG. Omg unified modeling specification, 2001. Version 1.4.

[16] OMG. Mda guide version 1.0.1, 2003. Document

Number: omg/2003-06-01 (2003).

[17] OMG. Submissions for mof 2.0 query/views/transformations request for proposal, 2003.

[18] OMG. Ontology definition meta-model, April 2004.

[19] Stephan Roser and Bernhard Bauer. Ontology-based model transformation. In *Proceedings of the ACM/IEEE 8th International Conference On Model Driven Engineering Languages And Systems (MoDELS/UML-2005) - Doctoral Symposium*, 2005.

[20] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, Vol.20(5):42–45, 2003.

[21] Miroslaw Staron and Ludwik Kuzniarz. Implementing uml model transformations for mda. In Kai Koskimies, Johan Lilius, Ivan Porres, and Kasper Osterbye, editors, *Proceedings of the 11th Nordic Workshop on Programming and Software Development Tools and Techniques NWPER'2004*, number 34 in General Publications, August 2004.

[22] W3C. W3c resource description framework, available at http://www.w3.org/rdf/.

[23] Annika Wagner. A pragmatical approach to rule-based transformations within uml using xmi.difference. In *Proeedings of WITUML 02: Workshop on Integration and Transformation of UML models*, Jul 2002.

[24] Yuxiao Zhao, Uwe Assmann, and Kristian Sandahl. Owl and ocl for semantic integration. Technical report, Programming Environmental Lab (PELAB), Department of Computer and Information Science, Linköping University, Sweden, May 2004.