Ontology Based Model Transformation Infrastructure

Arda Goknil[1], N. Yasemin Topaloglu[2]

Department of Computer Engineering, Ege University, Izmir, Turkey,
[1]goknil@staff.ege.edu.tr
[2]yasemin@bornova.ege.edu.tr

Abstract. Using MDA in ontology development has been investigated in several works recently. The mappings and transformations between the UML constructs and the OWL elements to develop ontologies are the main concern of these research projects. We propose another approach in order to achieve the collaboration between MDA and ontology technologies. We propose an ontology based model transformation infrastructure to transform application models by using query statements, transformation rules and models defined as ontologies in OWL. Using this approach in model transformation infrastructure will enable us to use semantic web and ontology facilities in model driven architecture. This paper will discuss how these two technologies come together to provide automatization in model transformations.

1 Introduction

Model Driven Architecture (MDA) is a recent approach that has been introduced by OMG [10]. MDA considers model generation as the core activity of software development and specifically, it aims to accomplish software development through generating *Platform Independent Models (PIMs)* and mappings these models to *Platform Specific Models (PSMs)*. The main idea behind this is to enable software developers to work in a higher abstraction layer than the code level. As a consequence, models become the primary artifacts of software development [8]. To define mappings between models, *model transformation*, which takes one or more source models as input and produces one or more models as output, according to a set of transformation rules is needed.

An ontology is 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 [9]. Web Ontology Language (OWL) is a technology for ontology development and knowledge representation in Semantic Web [2]. OWL defines and instantiates Web ontologies. Recent works discuss that UML [12] could be a key technology for the ontology development bottleneck [1] [5] [6]. 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 [13]. Alternatively to the established views, we propose another approach for the collaboration between MDA and OWL. While recent works discuss the contributions of MDA to ontology development, we discuss the possible contributions of ontologies to MDA. We propose an ontology based model transformation infrastructure to transform application models by using query statements, transformation rules and models defined as ontologies in OWL.

In this paper, we discuss our ontology based model transformation approach and define the ontologies for model transformations within the context of MDA. We base our proposal on the idea that the current technologies for model transformations are not enough for interoperability of the model queries and transformation rules. The recent popular technologies to identify transformation rules are XMI and XSLT [15].

The paper is organized as follows. In Section 2, we discuss the general characteristics and underlying concepts of the ontology based model transformations. In Section 3, we introduce our approach and define the ontologies for model transformation infrastructure. Section 4 includes the conclusions.

2 Overview of Ontology Based Model Transformations

2.1 Web Ontology Language (OWL)

Web Ontology Language (OWL) is a technology to provide a standard language for the representation of ontologies on the web. OWL is a result of the ongoing process of defining a standard ontology web language. It is an extension of *Resource Description Framework* (RDF) [17]. OWL provides a rich set of vocabulary to catch all the relationships between classes and properties. An OWL document can include an optional ontology header and any number of class, property, and individual descriptions or axioms.

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. Individuals are instances of classes, and properties may be used to relate one individual to another. These properties can be used to state relationships between individuals or from individuals to data values. For instance, an individual named *Olca* may be described as an instance of the class *Student* and the property *hasStudent* may be used to relate the individual *Olca* to the individual *EgeUniversity* which is derived from the class *University*. 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. Similar to object-oriented programming, class hierarchies may be created by using one or more statements which shows that a class is a subclass of another class [2]. For instance, the class *UniversityStudent* is the subclass of the class *Student*. OWL allows restrictions to be placed on how properties can be used by instances of a class. This restriction mechanism in OWL provides to define constraints, which can not be specified in UML or other modeling techniques.

2.2 General Concepts of the Model Transformation Ontology

Model transformation is the core activity in MDA 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 [14]. 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 [3][15][16].

XMI allows us encoding models in sets of XML tags to make them tool independent and interoperable. XSLT is another technology that enables to work on XML documents for model transformations. Though XMI and XSLT have reached wide usage, the interoperability and extendibility they provide are not sufficient. XMI is designed for interoperability among different case tools and it provides mechanisms for the exchange of UML models but it is not suitable for more structural interoperability.

The three main components of MOF 2.0 Query/Views/Transformations RFP [11] should be considered in the definition of model transformation ontology. The QVT RFP is issued by the Object Management Group (OMG) and seeks a standard solution for model manipulation. The three main subjects of model transformation defined by QVT [11]:

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

In our work, these three parts are defined as ontological. Defining queries and transformation in an ontology format will enable us to specify the structure of how meta entities and the relations between them are kept. Also queries defined in different transformation architectures will understand each other with the help of ontological approaches. To define instances from classes in XMI, you must define the meta classes in the same document. But in OWL, all instance queries reference a shared query ontology for the definitions of meta classes to define instances. The ontologies of these parts are defined as OWL documents. The definition in the OWL document provides a meta model for model transformations. For every instance transformation, instance ontologies can be derived from the meta ontologies.

3 Modeling the Transformation Components As Ontologies

As mentioned in QVT [11], the transformation infrastructure is constituted of three main structures as query, view and transformation. In our approach, we propose to model the meta entities and instances of these structures as ontologies.

3.1 Querying Application Models With Ontological Structures

Queries take a model as input, and select specific elements from that model. The aim is to detect the specific source and target patterns in application models. For that reason, different query languages have the same meta structures like selection and condition. These main structures are the basis of the query ontology.

Two different ontology documents are needed to query an application model. The first ontology document includes the meta classes of the query meta model. This meta model defines the main entities and the possible associations of these entities. The second document contains the instance query. The instance query selects the specific elements in the application document, and it is derived from the meta entities which are defined in the first query ontology. Figure-1 shows the relationship between the instance query ontology, meta query ontology, application model and the engine that process the query on the application model. *Query.rdf* includes the meta classes which constitute the meta model of model queries. These meta classes are the main selection elements like *Select, Where, And, Or, Not.* They associate the model elements to constitute the source and target patterns. *InstanceQuery.rdf* includes the instances of the classes in *Query.rdf* to define an executable query for an application model. *Query.rdf* is a kind of schema for query instances and defines the possible queries with its constructs.

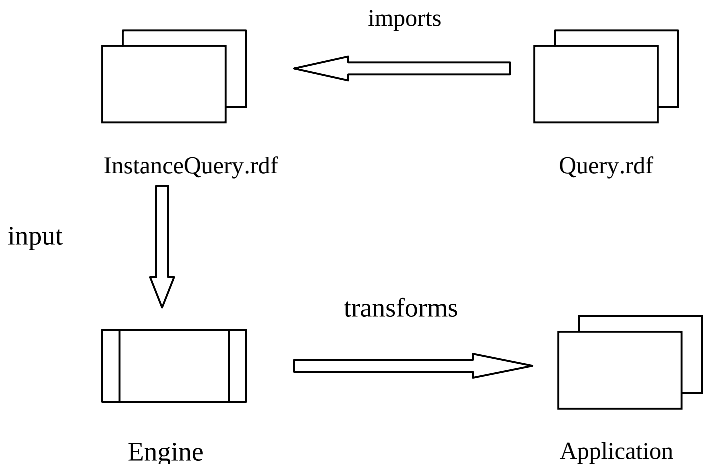

Figure-1. Deriving Query Instances from Query.rdf.

We propose a simple query language whose meta-model is shown in Figure-2[1]. The elements of this meta model constitute the structures in *Query.rdf*. The *Query* class in Figure-2 defines the query which is composed of two parts as *Where* and *Select*. The *Select* class associates with model elements which are derived after query

[1] Instead of showing the OWL document, we model our ontology definition by using UML class diagrams because of the space limitation in this paper.

processing. The *Where* class defines the condition in the query and is composed of the *Boolean terms (And, Or, Not), model elements* and *query references*.

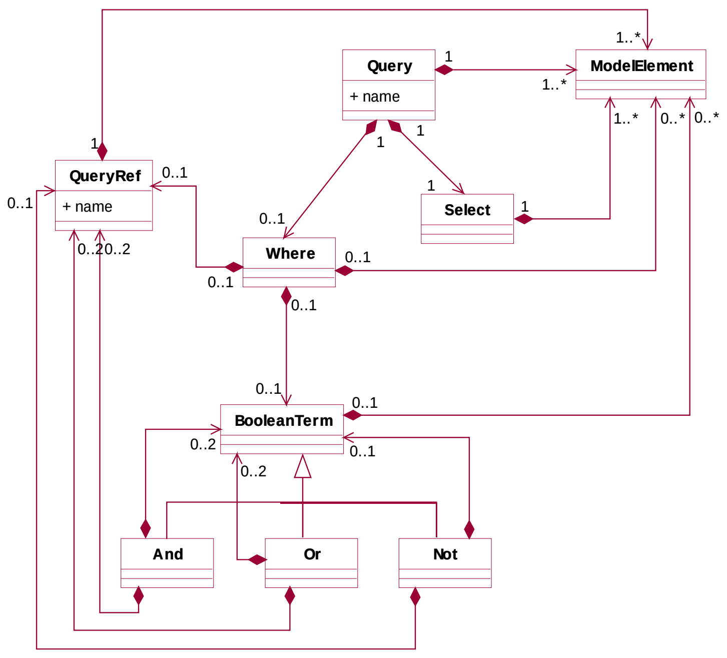

Figure-2. The Meta Classes in Query.rdf.

The *QueryRef* denotes other queries that are referenced in the *Where* term of the query. This enables us to use queries as recursive functions. The main difference between the *QueryRef* and the *Query* classes is that the *QueryRef* class is only a reference and does not contain the selection and the condition terms in the ontology where it is used. It defines the parameters which the *Query* it references uses in its own ontology. The Boolean terms include the model definitions as the conditions on the application model. The *Where* class may have these three classes in different combinations. The query ontology can limit the possible combinations that can be obtained from the meta model. The restrictions in the aggregation mechanism of the *Where* class and its collaborators can be defined in OWL as shown below:

```
<owl:Restriction>
      <owl:onProperty rdf:resource="#hasDefiniton">
        <owl:allValuesFrom>
             <owl:Class>
                  <owl:unionOf>
                    <owl:Class rdf:about="#QueryRef">
                    <owl:Class rdf:about="#BooleanTerm">
                    <owl:Class rdf:about="#ModelElement">
                  </owl:unionOf>
             </owl:Class>
```

```
         </owl:allValuesFrom>
       </owl:onProperty>
  </owl:Restriction>

  <owl:Restriction>
     <owl:onProperty rdf:resource="#hasDefinition"/>
    <owl:cardinality
  rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInt
eger">1</owl:cardinality>
  </owl:Restriction>
```

The property named *hasDefinition* defines the aggregation between the *Where* class and its collaborators, the *BooleanTerm* class, the *ModelElement* class and the *QueryRef* class. The first restriction defines that the *Where* class may have the *QueryRef*, the *BooleanTerm* and the *ModelElement* classes but with the second restriction it may have only one of them at once. The restriction mechanism in OWL provides to define constraints, which can not be specified in UML or other modeling techniques, for meta classes of our query model.

In Figure-2, we show the main classes that the query ontology must have. We can extend this ontology with additional structures for more complex model queries. For instance, there may be a set of same elements after the query processing. The query result set may have a model including a class associated with a set of same elements. To handle the set of model elements with iterations, there may be a container class to keep model elements as a set. A query which selects a class with a public attribute may return more than one public attribute of the class. We can handle the set of public attributes in the class in a set structure. Without a set structure, the query only matches the class with one public attribute at once. This set structure enables us to match one class with the set of its public attributes all at once.

Defining query models as ontologies allows us to extend this query meta model. The *InstanceQuery* ontologies are derived from *Query.rdf* for every model query like in Figure-1. In our work, *InstanceQuery.rdf* provides a query definition matching UML classes and their attributes, both owned by the class, and all of its superclasses. The query [4] shown below is an example of this.

```
  QUERY hasAttr(C, A)
  SELECT Class C, Attribute A, Class C2
  WHERE A.owner=C OR (C.super=C2 AND hasAttr(C2, A))
```

It is possible for ontologies to be treated as reusable 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 example, the *InstanceQuery* defining the *hasAttr* query imports *Query.rdf* to create individuals from the meta classes as shown below:

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

Every individual created in the *QueryInstance* references the class defined in the *Query* ontology. In our case, the *Where* individual has an *Or* individual. This *Or* individual has two properties named the *left-hand side* and the *right-hand side*. The left-hand side has a clause which defines *(A.owner=C)* and the right-hand side has an *And* individual. The *And* individual has *(C.super=C2)* clause in the left-hand side and a *QueryRef* referencing the *hasAttr* query with the parameters as *Class C2*, and *Attribute A*. Figure-3 shows this condition structure. In the ontology, we define *(A.owner=C)* clause with *Class C* which has *Attribute A*. Every model element used in the query is defined and is aggregated by the *Query* individual. We use their references while defining the clauses in the conditions. It means that the reference of the *Class C* has the reference of the *Attribute A*. The reference mechanism allows us to define conditions on model elements by using temporary clauses. The model elements defined inside the query are accessed through their references while the conditions are defined.

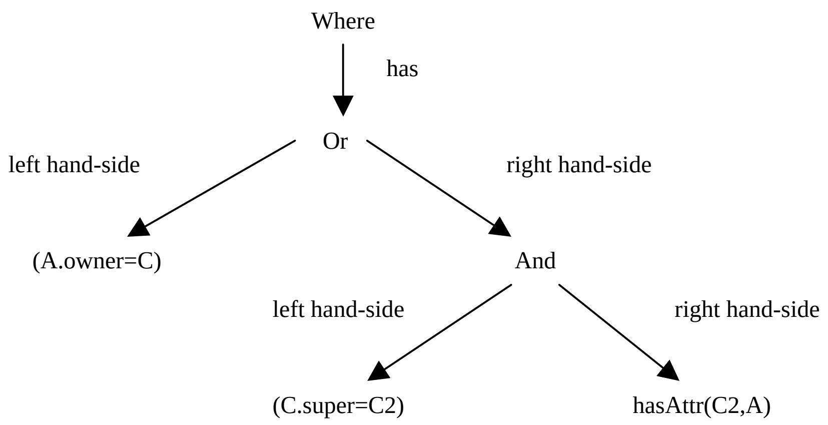

Figure-3. The Structure of the Condition Statement in the Instance Ontology.

3.2 Transforming Application Models With Ontological Structures

Transformations take a model as input and update it or create a new model. The submission for MOF 2.0 QVT RFP [11] split queries, views and transformations into two distinct groups. Queries and transformations may possibly create views, but views themselves are passive [11]. In our work, we consider that a transformation includes both queries and transformation operations. While queries select specific elements from the application model, transformation operations are applied to these selected model elements to transform the application model. The meta transformation ontology includes both the meta classes of transformation operations and queries. It can be considered that the transformation ontology is an extended query ontology to support the transformation operations.

The relationship between transformation ontology and transformations is similar to the relationship between meta ontology and instance ontology of queries that are discussed in Section 3.1. *Transformation.rdf* includes the meta classes which constitute the meta model of transformations and the instances in instance transformations are derived from these meta classes. *Transformation.rdf* is a kind of schema for transformation instances and defines the possible transformations with its constructs. Figure-4 shows the structures in our transformation ontology as a UML diagram.

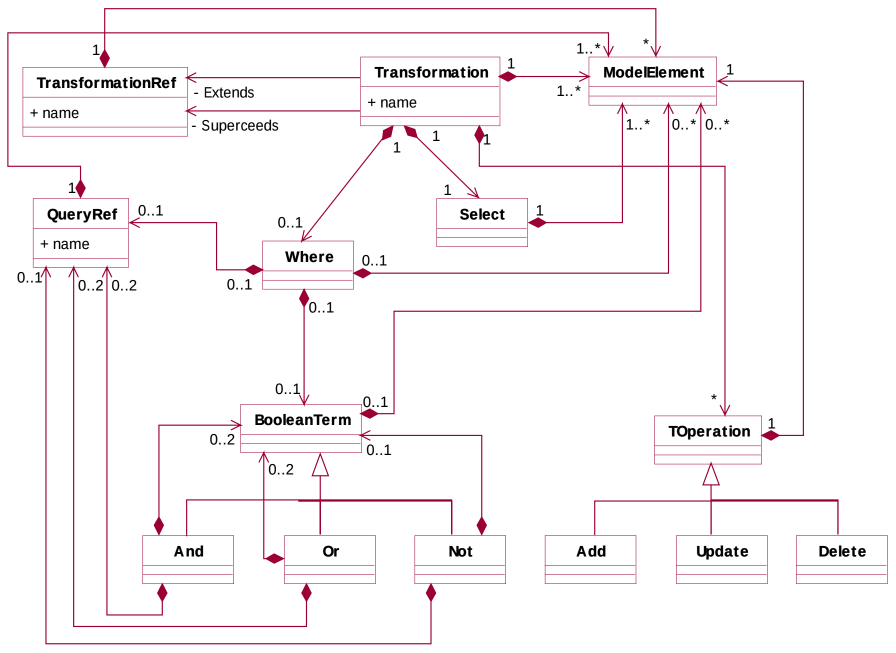

Figure-4. The Meta Classes in Transformation.rdf.

The *Transformation* class in Figure-4 defines the transformation itself. It has the *Select* and the *Where* classes like the *Query* class in *Query.rdf* because the transformation includes both model queries and transformation operations inside. The *Where* class is associated with the *QueryRef* class because a query instance can be referenced in the query condition of the transformation ontology. The query structures in *Transformation.rdf* and *Query.rdf* are the same. Transformations may be related to other transformations. [4] defines two ways for this relation: *Extends* and *Supersedes*. The associations between the *Transformation* and the *TransformationRef* classes in our ontology denote this relationship. Here, we have the OWL extensibility facilities to support other possible transformation relations in our ontology. Other possible relations between transformations can also be added to the transformation ontology in different approaches. The *TOperation* class and its sub-classes encapsulate the transformation operations on application models. The sub-classes of the *TOperation* class in our ontology are the *Add*, the *Update* and the *Delete* classes. We use them in our ontology to define the simplest operations. They are also the atomic operations and operate on meta class instances. In different and more complicated transformation approaches, more abstract and high-level classes may be used to define transformation operations.

Below, we give an example for a transformation definition which converts a public attribute from a given class to private and also creates the getter operation:

```
Transformation makingAttributePrivate(C, A)
SELECT Class C, Attribute A
WHERE (A.owner=C) AND (A.visibility=”Private”)
MAKE  A.visibility=”Private”
      Define getMet = new Operation()
      getMet.name=”getAttr”
      getMet.owner=C
```

The *Make* part after model query in the transformation defines the transformation operations on the application model. The first operation is making the visibility of *Attribute A* private. It is an update operation denoted by the individual derived from the *Update* class in the ontology. The second step in the transformation is creating a *get* method in *Class C.* An individual derived from the *Add* class does the creation of the get method named *getAttr* in *Class C.*

The third component of model transformations is views. Views are models that are derived from other models. Application models can also be defined in OWL instead of XMI where Case tools export and import the application models.

4 Conclusion

In this paper, we proposed an ontology based model transformation infrastructure. Ontologies provide a shared and common understanding of a domain. We consider the domain as model transformation in the context of model transformation languages and model the main constructs of model transformations. It enables us to extend our ontologies for future constructs of model transformations and allows communication of rules across applications. We used OWL in the definition of our ontologies because it is executable and is supported by tools. The restriction mechanism in OWL allows defining constraints about the instance models derived from meta models. OWL which provides to define assertions for UML has a precise semantics and is compared with *Object Constraint Language* (OCL) [18]. Some programmatic environments [7] include OWL APIs. These environments provide persistent storage, reading and writing OWL documents. Using OWL in model transformation infrastructure allows us to use the current semantic technologies to constitute the transformation engines. Loading and compiling the parts of model transformation are processed by the help of current ontology APIs.

Our aim is to investigate the possible contributions of ontologies to MDA and an ontology based model transformation infrastructure. We think that ontologies will play an important role in the development of MDA. In our future work, we will extend our transformation ontology with the constructs that support new transformation domains.

References

1. Backlawski, K. et al: Extending the Unified Modeling Language for Ontology Development. Int. J. Software and Systems Modeling, Vol.1, No.2 (2002) 142-156
2. Dean, M., Schreiber, G. (eds): OWL Web Ontology Language Reference W3C Recommendation, Feb 10, 2004, http://www.w3.org/TR/owl-ref/
3. Demuth, B., Obermaier, S.: Experiements with XMI Based Transformations of Software Models. WITUML'01, Genova Italy, April (2001)
4. Duddy, K., Gerber, A., Lawley, M., Raymond, K., Steel, J.: Model Transformation: A declarative, reusable patterns approach. In Proceedings EDOC 2003, pp 174-185
5. Falkovych, K., Sabou, M., Stuckenschmidth, H.: UML for The Semantic Web: Transformation-Based Approaches. In Knowledge Transformation in Semantic Web, IOS Press, Vol.95 (2003), pp 92-106
6. Gasevic, D., Djuric, D., Devedzic, V., Damjanovic, V.: Approaching OWL to MDA Through Technological Spaces. WISME@UML'2004, Lisbon Portugal, 2004
7. Jena: A Semantic Web Framework For Java. http://jena.sourceforge.net/
8. Judson, S., France, R., Carver, D.: Specifying Model Transformations at the Metamodel Level. WISME@UML'2003, San Francisco USA, October (2003)
9. Noy, N., McGuinnes, D.: Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, March (2001).
10. OMG: MDA Guide Version 1.0.1. The Object Management Group, Document Number: omg/2003-06-01 (2003)
11. OMG: Submissions for MOF 2.0 Query/Views/Transformations Request for Proposal. The Object Management Group (2003)
12. OMG: OMG Unified Modeling Specification. Version 1.4. (2001)
13. OMG: Ontology Definition Meta-Model, http://www.omg.org/cgi-bin/doc?ad/03-08-06 (Current Apr 3, 2004)
14. Sendall, S., Kozaczynski, W.: Model Transformation – the Heart and Soul of Model-Driven Software Development. IEEE Software Sep/Oct. (2003), pp. 42-45
15. Staron, M., Kuzniarz, L.: Implementing UML model transformations for MDA. NWPER'2004, Turku Finland, August (2004)
16. Wagner, A.: A pragmatical approach to rule-based transformations within UML using XMI.difference. WITUML 2002, Malaga Spain, June (2002)
17. W3C Resource Description Framework. http://www.w3.org/RDF/
18. Zhao, Y., Assman, U., Sandahl, K.: OWL and OCL for Semantic Integration. Technical Report, Programming Environmental Lab (PELAB), Department of Computer and Information Science, Linköping University, Sweden.