Towards using OWL DL as a metamodelling framework for ATL

Dennis Wagelaar[⋆]

Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium
dennis.wagelaar@vub.ac.be

Abstract. Ontologies have become increasingly relevant to the modelling community, providing a knowledgebase to support several software engineering activities. As such, several efforts have been made to integrate ontology technology with modelling technology. This has resulted in the Ontology Definition Metamodel (ODM), which allows model transformation languages to interact directly with ontologies. The ODM limits itself to data structure, however, and does not integrate the reasoning capabilities within ontologies. Therefore, we have done an experiment with an OWL DL driver for ATL, where ATL has direct access to the reasoning capabilities of the OWL DL ontology language. OWL DL is used as a metamodelling framework, where OWL classes serve as metaclasses and OWL individuals serve as model elements. The envisioned benefits are optimisation possibilities through automatic classification, and static analysis of ATL transformations written using OWL DL expressions. This paper will discuss our findings so far regarding these two envisioned benefits.

1 Introduction

Ontologies are becoming more and more prominent, and they can provide a supporting knowledgebase for several software engineering activities, such as domain engineering [1], semantic middleware [2], inconsistency management [3], platform dependency management [4], and several others [5]. As such, they are also gaining in relevance to the modelling community. This has resulted in the Ontology Definition Metamodel (ODM) [6][7], which provides a metamodel of the OWL ontology language, and allows model transformation languages to interact directly with ontologies. However, the ODM limits itself to the structural aspects of an ontology, and does not provide an interface for the reasoning capabilities for ontologies. Therefore, we have done an experiment with an OWL DL driver for ATL, which allows for direct access to the reasoning capabilities of the OWL DL language. The driver itself uses the OWLAPI ontology repository framework [8] and the Pellet ontology reasoner [9] to access ontologies and reason about them.

[⋆] The author's work is funded by a postdoctoral research grant provided by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT-Flanders)

In this experiment, OWL DL is used as a metamodelling framework, where OWL classes serve as metaclasses and OWL individuals serve as model elements. The OWL DL reasoner provides the inferred OWL class hierarchy, as well as the inferred classes for each OWL individual. Whereas in the modelling world OCL is used to express constraints, OWL DL provides its own language constructs for this purpose. The OWL DL language allows constraints – or restrictions – to be encapsulated *inside* (by subsumption) or even encapsulated *as* (by equivalence) an OWL class. An OWL DL restriction encapsulated as a class can automatically be classified in an inferred class hierarchy by the OWL DL reasoner, such that the OWL restriction is represented as a metaclass in the metamodel. The OWL DL reasoner also automatically infers the instances of this metaclass, which are those instances that conform to the OWL DL restriction.

The envisioned benefits of using OWL DL as a metamodelling framework are optimisation possibilities through automatic classification of OWL DL class expressions, and static analysis of ATL transformations written using OWL DL expressions. In the remainder of this paper, we will first discuss the development of the OWL4ATL driver, and our findings of mapping the ATL metamodelling constructs onto OWL DL constructs. Then, we will discuss our findings so far regarding the two envisioned benefits.

2 OWL4ATL

The ATL virtual machine allows for defining new *drivers* that implement ATL's model and metamodel abstractions. We've implemented such a driver for OWL DL[1], which maps metaclasses onto OWL classes and model elements onto OWL individuals. Table 1 shows how ATL's main language constructs for (meta-) model representation are mapped onto OWL language constructs, with the implementing Java classes in parentheses. ATL's primitive data types are mapped onto the closest XSD data types used by OWL.

ATL language construct	OWL language construct
Model (ASMModel)	OWLOntology (ASMOWLModel)
Metaclass (ASMModelElement)	OWLClass (ASMOWLModelElement)
Model element (ASMModelElement)	OWLIndividual (ASMOWLModelElement)
Reference (ASMModelElement)	OWLObjectProperty (ASMOWLModelElement)
Attribute (ASMModelElement)	OWLDataProperty (ASMOWLModelElement)

Table 1. ATL language construct to OWL language construct mappings.

[1] http://soft.vub.ac.be/viewvc/ATL/org.eclipse.m2m.atl.drivers.owl4atl/

2.1 Mapping metamodels to OWL

As the OWLAPI does not provide a metacircular implementation of OWLClass (where OWLClass is an instance of OWLClass) in the OWL DL language, the metametamodel must be implemented separately. This has not yet been done, and as a result only M2 models can be used as metamodels. This means that OWL ontologies of instances can be transformed, using OWL ontologies of classes as metamodels. As a consequence, we've used the Eclipse EODM[2] implementation of the ODM to generate an initial metamodel in OWL, consisting of OWL classes. As an example metamodel, we've used the Eclipse UML metamodel, as it covers all features of the EMF Ecore metamodelling language. The ATL transformation module used is called "ECORE2OWL.atl", the resulting metamodel "ontologies/UML.owl", which can both be found at http://soft.vub.ac.be/viewvc/ATL/OWLMatching/.

During the mapping of the UML metamodel to OWL, some of the semantic differences between Ecore and OWL became apparent:

- **OWL properties are defined within the namespace of the ontology, not the namespace of the OWL class on which one can define property values.** Whereas Ecore EAttributes and EReferences are defined within the namespace of a single EClass, OWL properties have a *domain* and a *range*. The domain specifies all OWL classes for which one can define property values, and the range specifies the valid types of the values. The domain of an EReference or EAttribute is always limited to a single EClass. When mapping only from Ecore to OWL, it is sufficient to add the name of the domain class to the property name to make sure the OWL property has a unique name within the ontology.
- **OWL properties do not support containment.** EMF models are in the form of a containment tree, where EReferences can be defined as containment references. OWL does not support the concept of containment, other than ontology elements being contained in an ontology (namespace). In order to retain the information of whether an OWL property represents a containment EReference, four "meta-properties" are introduced in the ontology: "allContainment", "allContainer", "containment", and "container". "allContainment" is a transitive property and represents the transitive closure of contained model elements. "allContainer" is a transitive property that represents the transitive closure of containing (parent) model elements. "containment" and "container" are non-transitive subproperties of "allContainment" and "allContainer", and represent the *direct* contained and containing model elements. All containment properties are represented as subproperties of "containment", and all container properties are represented as subproperties of "container". The OWL subproperty relationship means that each value of a subproperty can also be considered as a value of its superproperties. For example, any value of the UML "packagedElement" property is also a value of the "containment" and "allContainment" property. This

[2] http://www.alphaworks.ibm.com/tech/semanticstk

representation also allows for the implementation of the ATL "refImmediateComposite()" helper method.

2.2 Mapping models to OWL

Bridging the above semantic differences between Ecore and OWL is sufficient to create an OWL representation of the UML metamodel expressed in Ecore. It is also possible to copy a UML model based on Eclipse UML to an ontology based on the generated UML meta-ontology. Another, more fundamental semantic difference became apparent when trying to copy that UML ontology to another UML ontology:

– **OWL DL follows the Open World Assumption, whereas Ecore follows the Closed World Assumption.** This basically comes down to the principle that OWL DL assumes the available knowledge to be incomplete, whereas Ecore assumes it to be complete. The concrete issue that rises here is whether or not a model element can be considered an instance of a specific class. In Ecore, declaring a model element to be an instance of a metaclass is sufficient to know that it is *only* an instance of that metaclass (and its superclasses). In OWL DL, however, one must explicitly state that an individual is *not* an instance of a certain class. A partial solution for bridging this difference is to assert each new OWL individual as an instance of its OWL class, as well as of the complement of each subclass of that OWL class.

The ATL transformation rules that triggered this issue is the following:

```
rule Model {
  from s : UML2!Model
  to t : UML2!Model (
    ...)
}

rule Package {
  from s : UML2!Package (s.oclIsTypeOf(UML2!Package))
  to t : UML2!Package (
    ...)
}
```

The Model rule copies all instances of the "Model" metaclass, and Package copies all instances of "Package". The Package rule explicitly checks that it does not accidentally copy "Model" instances, as "Model" is a subclass of "Package" in the metamodel, and would otherwise trigger the Package rule. In OWL DL, any instance `s` of "Package" must explicitly be marked as *not* an instance of "Model" for the reasoner to conclude that `s.oclIsTypeOf(UML2!Package)` is false.

A complete solution to the Open World vs. Closed World Assumption problem would require additional "closing" assertions to be added to the OWL representation of the metamodel:

– **Each pair of OWL classes that do not have a common sub- or superclass must be declared disjoint.** This enforces that OWL individuals are considered *not* to be instances of other classes than its asserted class, or superclasses of its asserted class. This follows the general semantics of metamodelling languages such as Ecore. This cannot be done statically, as Ecore models and OWL ontologies can both be extended with new classes. Multiple inheritance may cause a pair of classes that previously had no common subclasses to now have common subclasses. Currently, OWL4ATL does not address this issue.

2.3 Mapping OCL expressions to OWL restrictions

Up to this point, the experiment has remained limited to OWL class/property expressions that reflect Ecore expressions. ATL uses OCL to navigate over metamodels and express restrictions on **from** statements. OWL DL provides its own language constructs for expressing restrictions. Consider the following example ATL rule, which transforms all UML Properties that are public:

```
rule PublicProperty {
  from s : UML2!Property (
    s.oclIsTypeOf(UML2!Property) and s.visibility = #public)
  to t : UML2!Property (
    ...)
}
```

In OWL DL, this OCL restriction can be written as an OWL class expression in extended OWL Manchester syntax:

```
PublicProperty subClassOf Property
PublicProperty equivalentWith (
  (Property_visibility value "public") and not ExtensionEnd and not Port)
```

This means that PublicProperty is the set of all Property instances that have Property_visibility set to “public” and are not an ExtensionEnd or a Port (the subclasses of Property). As this expression is encapsulated as an OWL class, it can be used from ATL as follows:

```
rule PublicProperty {
  from s : UML2!PublicProperty
  to t : UML2!Property (
    ...)
}
```

The OWL DL reasoner also automatically classifies OWL class expressions in the inferred class hierarchy. Consider the following ATL rule and its OWL representation:

```
rule AllProperty {
  from s : UML2!Property (
    s.oclIsTypeOf(UML2!Property))
  to t : UML2!Property (
    ...)
}

AllProperty subClassOf Property
AllProperty equivalentWith (not ExtensionEnd and not Port)
```

The OWL DL reasoner would classify PublicProperty as a subclass of AllProperty, which in turn is a subclass of Property. Based on the inferred OWL class hierarchy, we can derive whether there is any overlap between the **from** parts of ATL rules. Overlap is not allowed between separate matched rules, while overlap is required between a super- and subrule in ATL.

3 Discussion

This paper has presented an experiment with using OWL DL as a metamodelling framework for ATL. The experiment is still in progress, but some results are already available. First of all, the development of the OWL4ATL driver that enables the use of OWL DL within ATL has uncovered several semantic differences between a regular metamodelling language, such as Ecore, and OWL DL. Two of these differences are structural differences, regarding namespaces and containment of elements, and can be bridged without many problems. Another difference is the Open World vs. Closed World Assumption, which is a fundamental difference in language semantics. The result is that complex mappings are required between regular metamodelling and OWL DL in order to represent the same modelling language. A start has been made with providing these mappings, but more mappings are required. Some of these mappings are also dynamic, and can change as the metamodel is extended. Whether or not these mappings are needed, depends on whether one wants to represent a regular metamodel in OWL DL, or whether one just wants to use an existing OWL DL ontology as a metamodel (and retain the Open World Assumption).

The first envisioned benefit of using OWL DL as a metamodelling framework is the possibility to optimise through automatic classification of OWL DL class expressions. We have demonstrated how OWL DL class expressions can be used in the **from** part of an ATL rule. ATL currently relies on caching of helper attributes to achieve a performance speedup. This only works if the developer uses helper attributes in the right places. By using OWL DL class expressions, the OWL DL reasoner can use its inferred class hierarchy to make smart assumptions about the instances of the classes. A subclass will have no more instances than its superclass, for example. We currently don't know if the OWL DL reasoner can be more efficient in finding class instances that ATL using helper attribute caching. We do know that current OWL DL reasoners have a limit in how many class equivalence expressions in combination with property restriction expressions they can handle, which lies somewhere below 1000. This puts a limit on the complexity of the transformation module.

The second envisioned benefit is the ability to perform static analysis of ATL transformations written using OWL DL expressions, similar to what is currently done in graph transformation using critical pair analysis [10] or in OCL using constraint programming tools [11]. Using the inferred OWL class hierarchy, it is possible to determine whether there is overlap between two class expressions. In addition, the OWL DL reasoner will be able to determine whether a class expression is satisfiable at all (a non-satisfiable class expression cannot have any instances). As long as we can translate an ATL **from** part expression to an OWL class expression, it is possible to perform this kind of static analysis.

As soon as recursive helper methods are used, however, a translation to an OWL DL expression is no longer possible. Currently, only a few examples have been translated into OWL DL expressions. An exhaustive translation of ATL expressions to OWL DL expressions will have to be developed. As most of the work consists of translating OCL to OWL DL, the work of Cabot et al., amongst other presented in [11], may serve as a basis here.

Acknowledgements

The author would like to thank Ragnhild Van Der Straeten, Ulrike Sattler, and Carsten Lutz for their discussion on mapping regular MOF/Ecore style metamodelling languages onto OWL DL, and using OWL DL as a metamodelling framework in a model transformation context.

References

1. Musen, M.A.: Domain ontologies in software engineering: use of Protégé with the EON architecture. Methods of Information in Medicine **37** (1998) 540–550
2. Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D.L., Sirin, E., Srinivasan, N.: Bringing Semantics to Web Services with OWL-S. World Wide Web **10** (2007) 243–277
3. Van Der Straeten, R., Jonckers, V., Mens, T.: A formal approach to model refactoring and model refinement. Software and Systems Modelling **6** (2007) 139–162
4. Wagelaar, D., Van Der Straeten, R.: Platform Ontologies for the Model-Driven Architecture. European Journal of Information Systems **16** (2007) 362–373
5. Happel, H.J., Seedorf, S.: Applications of Ontologies in Software Engineering. In: International Workshop on Semantic Web Enabled Software Engineering (SWESE'06). (2006)
6. Bézivin, J., Devedžić, V., Djurić, D., Favreau, J., Gašević, D., Jouault, F.: An M3-Neutral infrastructure for bridging model engineering and ontology engineering. In: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05) Geneva, Switzerland, Springer-Verlag (2005) 159–171
7. Object Management Group, Inc.: Ontology Definition Metamodel. (2006) Sixth Revised Submission to OMG/ RFP ad/2003-03-40, ad/2006-05-01.
8. Horridge, M., Bechhofer, S.: The OWL API: A Java API for Working with OWL 2 Ontologies. In: OWLED 2009, 6th OWL Experienced and Directions Workshop. (2009)
9. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics **5** (2007) 51–53
10. Mens, T., Taentzer, G., Runge, O.: Detecting Structural Refactoring Conflicts Using Critical Pair Analysis. Electr. Notes Theor. Comput. Sci. **127** (2005) 113–128
11. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Analysing Graph Transformation Rules Through OCL. In Vallecillo, A., Gray, J., Pierantonio, A., eds.: Proceedings of the First International Conference on Theory and Practice of Model Transformations (ICMT 2008), Zürich, Switzerland. Volume 5063 of Lecture Notes in Computer Science., Springer-Verlag (2008) 225–239