UML for the Semantic Web: Transformation-Based Approaches

K. Falkovych[1] M. Sabou[2] H. Stuckenschmidt[2]

[1] Department of Multimedia and Human Computer Interaction
CWI, Kruislaan 413 P.O. Box 94079 1090 GB
Amsterdam The Netherlands
Kateryna.Falkovych@cwi.nl

[2] Artificial Intelligence Department
Vrije Universiteit, De Boelelaan 1081, 1081HV,
Amsterdam, The Netherlands
{marta,heiner}@cs.vu.nl

Abstract. The perspective role of UML as a conceptual modelling language for the Semantic Web has become an important research topic. We argue that UML could be a key technology for overcoming the ontology development bottleneck thanks to its wide acceptance and sophisticated tool support. Transformational approaches are a promising way of establishing a connection between UML and web-based ontology languages. We compare some proposals for defining transformations between UML and web ontology languages and discuss the different ways they handle the conceptual differences between these languages. We identify commonalities and differences of the approaches and point out open questions that have not or not satisfyingly been addressed by existing approaches.

1 Introduction

The so-called Semantic Web [3] aims at enriching the World Wide Web with semantic information to enable systems to access and use information more efficiently. For this purpose a number of annotation languages have been developed in order to annotate information resources with content-related meta-data. In order to create, interpret and compare meta-data annotations, ontologies, explicit definitions of the vocabulary used in an information source, are needed. While meta-data can often be generated from the content of an information source, the development of ontologies is likely to become the major bottleneck in scaling up semantic annotations. Overcoming the modelling bottleneck requires a large number of professional ontology builders equipped with powerful modelling tools. The current situation is far from the required one. Today, ontologies are built by a small number of people, in most cases researchers, with prototype tools that provide some basic functionality for editing and storing ontological models. These tools have little in common with professional development environments we know from the area of software engineering. Admittedly, the tool support for ontological modelling has improved over the last years and editors are available now that support consistency checking, import of existing ontologies and visualization of ontologies, but at the moment it is unrealistic to claim that we can give these tools to non experts in ontological modelling and expect good modelling results.

Recently, the Unified Modelling Languages (UML) has been identified as a way of providing a partial solution to the modelling bottleneck. Being the standard modelling language in software engineering, UML has received wide attention not only in academia, but also in professional software development. As a consequence, UML is much better supported in terms of tools and available expertise than the emerging semantic web languages. The wide acceptance of UML makes it an ideal language to be used by a critical mass of people to build high quality models of information semantics for the semantic web. A number of researchers have recognized the importance of UML with ontological modelling, especially for the semantic web (e.g. [5]). In this paper, we discuss approaches, that are based on the idea of transforming domain models between UML and semantic web languages, focusing on the actual transformation between the two languages.

In the following, we contrast the different ideas behind UML and the semantic web languages concluding the resulting differences between these languages. Based on this high level discussion, we turn our interest to transformations between these languages. We first identify two slightly different use cases of employing transformations between UML and semantic web languages. Afterwards, we outline some existing approaches and their specific solutions to the transformation problem. After a discussion of the different choices made by different approaches we identify a number of open issues with respect to transformation that are essential for using UML on the semantic web. We conclude with an outline of a research agenda for an effective use of UML on the semantic web.

2 UML and Semantic Web Languages

The common feature of all transformation-based approaches to using UML on the semantic web is the fact that they focus on the languages as such. Other important aspects such as methodologies, modelling tools or reasoning support are assumed to be addressed by the corresponding languages communities. In fact, the standardization of UML on the one and OWL on the other hand has stimulated many developments around these languages that we can take advantage of once we have transformed a model into the respective language. Consequently, the central question in transformation-based approaches is concerned with the relation between UML and the semantic web languages. In order to understand the technical differences, it is important to recall a little bit of the history and the motivation that underlies the two languages.

2.1 Unified Modelling Language UML

Aim: The UML language is designed in order to integrate competing proposals for modelling languages in the area of software engineering. This integration effort was undertaken in order to push object-oriented design methods into industrial practice by providing a well-supported standard method that makes development processes more transparent.

Principles: UML is primarily designed as a language to be used by humans to document and communicate software designs. As a consequence, the main notation of UML is defined in terms of a graphical model rather than a formal language. In order to capture formal constraints, an additional constraint language (OCL) has to be used. Being an integration of different pre-existing proposals, UML is rather the union of these approaches than an intersection and aims at maximal expressivity. In order to cover all aspects of a system, the language uses different interacting diagram types each covering a specific aspect of the overall system.

Semantics: The fact that UML consists of different diagram types that interact with each other makes it difficult to define a uniform semantics. The only approach to define the semantics of UML that covers all diagrams is the meta-model approach. In this approach the modelling elements of the language are defined in terms of UML class diagrams and OCL constraints. Further, there are attempts to define formal semantics of parts of the language, such as class diagrams or state charts. Validation and transformation methods for UML models have been defined on the basis of these partial semantics.

In order to capture the various aspects of complex software systems, UML consists of not less than twelve different diagram types (according to OMG) each being designed to describe the system from a specific point of view. Four diagrams describe the static application structure, five different aspects of the dynamic behavior, and three represent ways to organize and manage the application modules. The class diagrams belong to the first category and they have been in the center of attention because there exists a direct relation between their elements and the parts of an ontology (classes, hierarchies, class attributes and axioms). Some language features, especially from the system structure part will be mentioned in the description of the transformation approaches in the following section. For a complete description of UML we refer to the official specifications [14].

2.2 Web Ontology Language OWL

Aim: The goal of the Web Ontology Language OWL is to provide a standard language for the representation of ontologies on the World Wide Web. Such a standardized language will support the development of a web-based ontology infrastructure by providing editors, storages, inference engines and meta-data annotation tools.

Principles: In order to fit into the general web architecture, OWL adopts a number of principles, including a XML-based encoding and backward compatibility with RDF Schema, the current W3C standard for conceptual modelling. As such, the language is designed to be handled by systems rather than by human users. The provision of a well founded logical semantics is a design principle of OWL. Beyond this, the developers of OWL follow the principle of minimality: rather than including as many modelling features as possible, the OWL language is restricted to a set of features that make logical reasoning feasible.

Semantics: OWL has a well founded semantics in the spirit of description logics [7], which is established by an interpretation mapping into an abstract domain. The language is very close to a specific description logic called $\mathcal{SHOIQ}$. Existing sound and complete reasoning procedures for this logic can be used to provide reasoning support for OWL models.

As a result of the ongoing process of defining a standard ontology web language, a number of intermediate versions of the language have been defined. These languages differ in some features, but their underlying design principles are similar. As different transformation approaches refer to different language versions, we briefly outline the development at this point. For details about differences between the languages, we refer to the original language specifications.

OIL The Ontology Inference Layer OIL [10] has been developed in the On-to-Knowledge project [11] as a language to support logical reasoning with and about ontologies. The principle behind the language is to have different levels of expressiveness where the lowest level is equivalent to RDF schema.

DAML The DARPA Agent Markup Language DAML [12] was developed in parallel with OIL. The goal was to provide a common basis for the DAML project. DAML primarily originated from the area of frame-based knowledge representation systems and hence it is less influenced by formal logic than OIL.

DAML+OIL The name DAML+OIL resulted from the decision to create a joint language on the basis of existing DAML and OIL. The language replaced DAML as the standard language of the DAML project and was defined in an official committee. Two versions of DAML+OIL exist, the March 2001 version [17] being more widely supported than the December 2000 version [16].

OWL The Web Ontology Language OWL [6] is the label under which DAML+OIL undergoes the process of becoming a W3C recommendation. OWL is based on DAML+OIL with different levels of expressivity included in the language. There exist two versions of the language: OWL Lite that includes restricted vocabulary sufficient to satisfy primarily user needs and OWL Full with complete OWL vocabulary.

2.3 Conclusions

By comparing the natures of the two languages we can draw two conclusions. On the one hand they complement each other. While UML is designed for model building by human experts, OWL is designed to be used at run time to provide guidance for intelligent processing methods. This complementary character justifies the idea of combining OWL and UML in order to overcome the acquisition bottleneck. On the other hand, we can see that the translation is less than trivial, because of the differences between the two languages. The first challenge is to identify corresponding elements in the two languages, which will sometimes be difficult, because UML is biased by the later implementation. Second, we have to make sure that translations are backed by the semantics of the languages.

3 Existing Transformation Approaches

The benefits of using UML and DAML+OIL together have led to research where (1) *UML is used as a modelling syntax for knowledge representation languages such as DAML [1]*, or (2) *standard UML models are transformed into OIL or DAML+OIL ontologies [9], [8]*.

The first direction arose due to the fact that current ontology representation languages lack representation possibilities such as a graphical front-end. As a recently emerged language, DAML does not have sufficient tool support, while UML is a commonly accepted graphical notation that can be used to build DAML ontologies. It was discussed in the previous section that UML and ontology representation languages have certain incompatibilities.

These incompatibilities prevent the representation of all elements from the ontology representation languages with the help of UML. In order to avoid this a number of extensions to UML have been proposed.

The second direction addresses the problem of reusing previously specified knowledge. UML models cannot be easily exchanged over the Web, the reasoning possibilities with UML models being also quite restricted (see our arguments in section 4.2). To overcome these difficulties [8] and [9] propose transformations between UML and ontology representation languages.

These two directions induced the need to generate transformation rules between UML and ontology representation languages. Differently aimed matching between languages caused differences in the proposed mappings. While the first direction requires UML to be suited for representing DAML ontologies, the second one addresses the question of how UML concepts can be represented with an ontology representation language while preserving their semantics as much as possible. In the rest of this section we introduce these approaches and discuss transformation issues in more detail.

3.1 UML-based Tools for DAML

The first direction is represented by the work of Baclawski [1], where the relationships between UML and DAML have been investigated in order to provide support for visualizing complex ontologies and managing the ontology development process. Baclawski proposes a mapping between the two languages and gives a solution to the one of the most problematic differences between the languages. The mapping is done from the UML point of view while assuming that UML is used to represent DAML ontologies. As a consequence of this assumption a number of the UML class diagram elements are not presented in the mapping table. Although the paper focused at DAML, it remains valid for DAML+OIL.

It was argued in all related investigations that the biggest obstacle on the way to map UML and ontology representation languages is the notion of `Property`. From the first glance the notion of `Property` in knowledge representation languages corresponds to the notion of `association` in UML. `Property` is a first-class modelling element in all web-based knowledge representation languages starting from RDF(S). This means that `Property` can exist as an independent element of a language without being attached to other constructs (such as classes). At the same time UML associations are connected to classes with association ends. An association cannot exist without explicit connections to classes. It also means that each association is unique. Even if two associations have the same name they are connected to different classes and thus they are considered to be different. Thus, UML associations express local restrictions on instances of classes, while DAML properties in combination with other elements can express local as well as global restrictions. An association has only one domain and range, DAML `Property` can have more than one domain class. This makes it difficult to propose a mapping that is correct at the conceptual level. Different approaches have been proposed to tackle this problem, as discussed below.

To handle the difference between the notions of UML `association` and DAML `Property` Baclawski proposes an extension to the UML meta-model [1]. He suggests to enrich the Meta-Object Facility (MOF) specification [15] with the notions of `Property` and `Restriction` as it is shown in Figure 1.

Both `Property` and `Restriction` are modelled on the diagram as UML

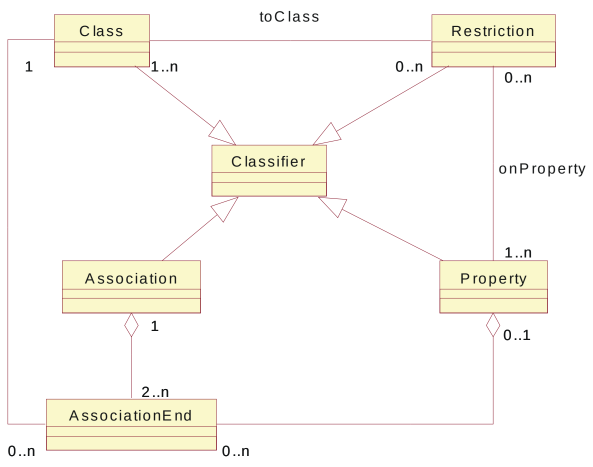

Figure 1: UML extension for property restrictions

`Classifiers`, enabling them to become first-class modelling primitives in UML. Also `Property` is modelled as an aggregation of zero or more association ends, allowing it to exist without being connected to a class. Each association end can be described by at most one `Property`.

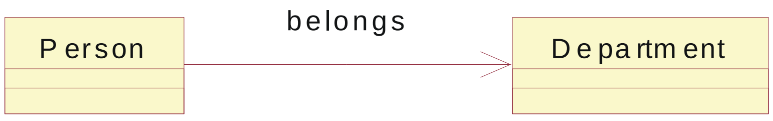

Figure 2: DAML property restriction as UML diagram

According to the diagram, the notions of `Property` and `Association` are overlapping to some extent, however `Property` relates to a DAML element. `Property` can be constrained by zero or more `Restrictions`. A `Restriction` can be related to one or more classes. This corresponds to a DAML construction of a property restriction to a class. Consider the diagram in Figure 2. It represents a restriction on the property `belongs` to the class `Department`.

By applying UML to DAML mapping the DAML translation of the UML diagram will result in a section of an ontology, depicted in the following. This example gives the short scenario of how the transformation works.

```
<daml:Class rdf:ID="Person">
    <rdfs:label>Person</rdfs:label>
    <rdfs:subClassOf>
        <daml:Restriction >
            <daml:onProperty rdf:resource="#belongs"/>
            <daml:toClass rdf:resource="#Department"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:Class rdf:ID="Department">
    <rdfs:label>Department</rdfs:label>
```

```
</daml:Class>

<daml:Property rdf:ID="belongs"/>
```

The inclusion of the proposed extension into a new UML 2.0/MOF 2.0 specification will have an impact on existing tools, but the authors believe that this impact will not be much bigger from the one anyway caused by the new specification. Moreover, it can lead to the general acceptance of UML as a modelling tool for ontologies.

3.2 Transformations of Standard UML Models into Ontology Representation Languages

We discuss two representative approaches in this direction: the first proposes a mapping that supports ontology extraction from UML diagrams; the second reports on using UML diagrams for modelling domain specific knowledge.

Extracting DAML+OIL Ontologies from UML Diagrams Falkovych [8] proposes a transformation of UML diagrams into DAML+OIL ontologies with preserving the semantics of UML concepts. This work emerged from an observation that there exist large sources of ontological knowledge already available in design documents of existing applications. Ontology extraction from UML diagrams and converting them into a powerful ontology representation language would enable reasoning about these ontologies and would allow knowledge interchange between heterogeneous environments. Hence, the main contribution of [8] lays in introducing a way of translating UML class diagrams into RDF(S) and DAML+OIL. Since RDF(S) can be considered as a subset of DAML+OIL, the UML to RDF(S) transformation is not discussed here.

All the issues about the differences between UML and DAML (DAML+OIL) discussed in Baclawski's work [1] are also relevant here. But the purposes of the work force us to look from the different perspective at the transformation task. The mapping is proposed for every UML element, which is possible to map into DAML+OIL constructs. For the complete mapping table the reader is referred to [8]. In order to have better understanding about the specificity of the mapping, the main issues are discussed below.

Since association is unique in UML, it is mapped into `daml:ObjectProperty` with a unique identifier for the property name. This makes it possible to distinguish different associations with the same name as different DAML+OIL properties.

UML attributes are mapped in a similar way, since each attribute has a local scope within its class. An attribute is mapped into `daml:DatatypeProperty` with a unique identifier attached. The rationale of mapping an attribute only to `daml:DatatypeProperty` and not to `daml:ObjectProperty` is that usually the type (range) of an attribute is a data type. Although in UML an attribute can have an object as its type, this situation occurs infrequently. Thus, in this mapping the assumption was made that all properties that have an object as a range are modelled as associations and not as attributes.

In order to preserve semantics of UML associations and to distinguish association ends, a taxonomy of association types is introduced (Figure 3). The notion of an association in the taxonomy is divided into four subtypes, namely binary association, unidirectional association, aggregation and composition, which are decomposed further into specific subtypes introduced for the mapping purposes. At the lowest level of decomposition, which is not present in the figure, all association types can have a name or be unnamed and they can have

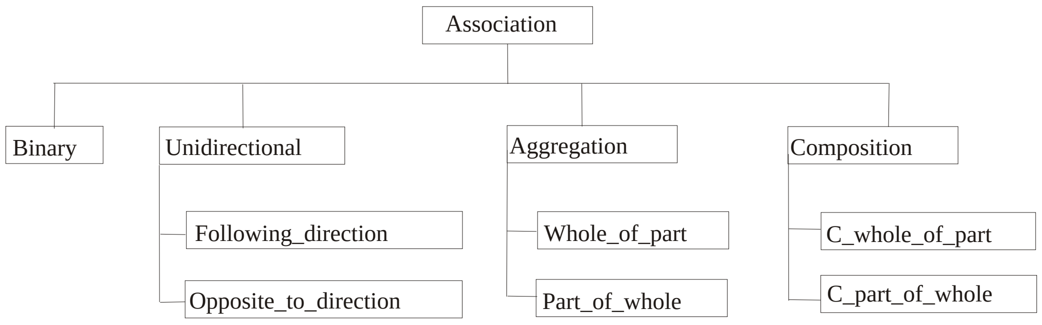

Figure 3: Taxonomy of association types

role names attached to association ends. All associations in a particular diagram are modelled as sub-properties of corresponding association types from the taxonomy.

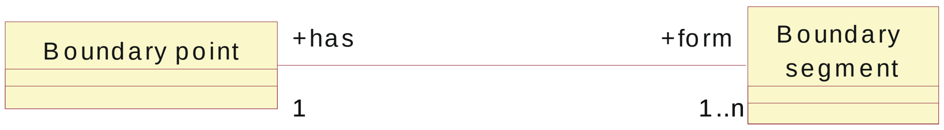

Figure 4: Unnamed binary association with role names and multiplicity constraints

A binary association is navigable in both ways, thus it is mapped into two `daml:ObjectProperty` elements, which are inverse (`daml:inverseOf`) of each other. These two properties have a name of an association with the unique identifier attached to it (or just an identifier that serves as a property name in the case of an unnamed association) and they are distinguished by adding an underline symbol ('_') to one of them. The need to distinguish them comes from the need to map possibly present role names and multiplicity constraints attached to the ends of an association. Roles give some additional meaning about roles of classes in a relation, thus a role is specified as `rdfs:subClassOf` of an association it is attached to. Cardinality constraints are specified for associations as well as for roles. The example below clarifies these issues.

```
<daml:Class rdf:ID="Boundary point">
    <rdfs:label>Boundary point</rdfs:label>
    <rdfs:subClassOf>
        <daml:Restriction daml:minCardinality="1" >
            <daml:onProperty rdf:resource="#has_G.4"/>
            <daml:toClass rdf:resource="#Boundary segment"/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:minCardinality="1" >
            <daml:onProperty rdf:resource="#_G.2"/>
            <daml:toClass rdf:resource="#Boundary segment"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:ObjectProperty rdf:ID="has_G.4">
```

```
    <rdfs:subPropertyOf rdf:resource="#_G.2"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="_G.2">
    <rdfs:subPropertyOf rdf:resource="#binary_unnamed"/>
</daml:ObjectProperty>

<daml:Class rdf:ID="Boundary segment">
    <rdfs:label>Boundary segment</rdfs:label>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1" >
            <daml:onProperty rdf:resource="#form_G.3"/>
            <daml:toClass rdf:resource="#Boundary point"/>
        </daml:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <daml:Restriction daml:cardinality="1" >
            <daml:onProperty rdf:resource="#G.2"/>
            <daml:toClass rdf:resource="#Boundary point"/>
        </daml:Restriction>
    </rdfs:subClassOf>
</daml:Class>

<daml:ObjectProperty rdf:ID="form_G.3">
    <rdfs:subPropertyOf rdf:resource="#G.2"/>
</daml:ObjectProperty>

<daml:ObjectProperty rdf:ID="G.2">
    <rdfs:subPropertyOf rdf:resource="#binary_unnamed"/>
</daml:ObjectProperty>
```

A unidirectional association is mapped in the same way with the difference that it is navigable in one way and thus two sub-properties of this association are distinguished (`Following_direction` and `Opposite_to_direction`). These sub-properties are not the inverse of each other. The same issues apply to aggregation and composition where different sub-property names are used to distinguish two ways of readability.

Domain-Specific Transformations An increasing interest in distributed problem solving for configuration tasks arose the need to represent configuration services as Web Services. To deal simultaneously with multiple service providers over a network and to interact with other configurators a commonly used representation formalism is required. Semantic Web languages such as OIL or DAML+OIL seem to be good candidates for such formalism. At the same time configuration system modelling requires a participation of domain experts and knowledge engineers. Knowledge representation languages are difficult to use during the modelling stage. The work of Felfernig [9] provides a set of rules to automatically transform configuration models represented in UML into an OIL representation. As such this research corresponds to the second direction of existing transformation approaches. UML elements for which the mapping is proposed constitute a configuration ontology. The set of elements required for UML configuration profile consists of classes, generalizations, associations and aggregations (part-whole relationships). Compatibilities and requirements are introduced through the stereotype associations `incompatible` and `requires`. Resources are described by a stereotype `Resource` with `produces` and `consumes` dependencies to component types. The mapping of basic language constructs is similar with other approaches. The specific issue, which we would like to discuss here, is a way of treating UML part-whole relationships in OIL. The example below illustrates this (Figure 5):

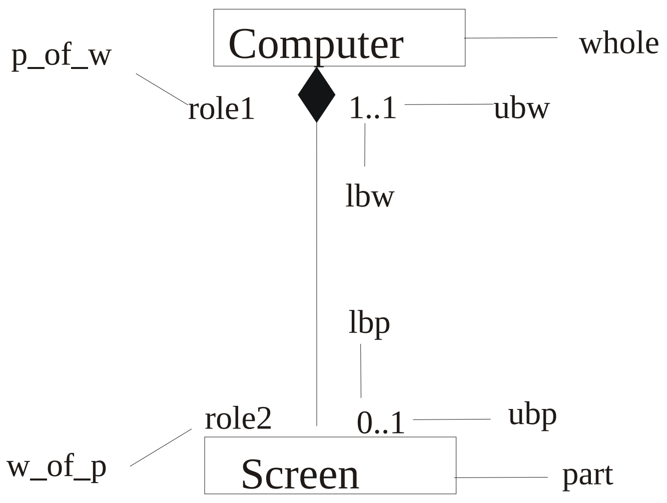

Figure 5: Graphical representation of part-whole relationships (taken from [9])

```
slot-def w-of-p
    subslot-of Partofcomposite
    inverse p-of-w
    domain p
    range w.
slot-def p-of-w
    subslot-of haspart
    inverse w-of-p
    domain w range p.

p:slot-constraint w-of-p min-cardinality lbw w,

p:slot-constraint w-of-p max-cardinality ubw w,

w:slot-constraint p-of-w min-cardinality lbp p,

w:slot-constraint p-of-w max-cardinality ubp p,
```

Two classes involved in a relation are distinguished as ‘whole’ and ‘part’. Roles attached to the part-whole relation are denoted with `part-of-whole (p-of-w)` and `whole-of-part (w-of-p)` names. Multiplicities are represented through lower and upper bounds of part and whole. Aggregation and composition are considered being disjoint. An additional restriction in OIL is specified to indicate that a ‘part’ component that is connected to a ‘whole’ component through a composition cannot be connected to any other component. The `source` and `target` classes of a relation are mapped into the `domain` and `range` OIL elements that occur due to the restrictive abilities of OIL with respect to DAML+OIL. The distinctive feature of DAML+OIL is the ability to define local constraints of domain and range, which correspond more to the UML notion of an association (see section 3.1).

4 Discussion

4.1 Summary of the Comparison

After introducing the main features of the approaches in the sections above we would like to underline differences and similarities between them.

Differences: The main difference between the approaches is the way they treat the semantics of UML.

Baclawski aims at providing a presentation syntax for DAML. He uses stereotypes in the mapping and proposes the extension to UML meta-model to make UML more expressive. The extension closes the gap between the two languages and makes UML more suited for representing DAML ontologies.

Falkovych approaches the problem in a different manner. The semantics of different kinds of associations are included in a taxonomy, which is then explicitly transformed into a DAML+OIL ontology. Every property that corresponds to a certain kind of an association in a UML diagram is defined as a sub-property of a corresponding association type from the taxonomy. In this way the semantics of associations is preserved after performing the transformation from one language to another. Being part of the resulting ontology, they can be used for example to ask for the relation between certain entities.

In the Felfernig work we saw an application of a transformation task to the configuration domain. His research shows a specific way of treating UML diagram elements representing a configuration model in OIL. The problem of the correspondence between UML association and DAML property is introduced here through the example of part-whole relationships. Since part-whole relationships are important for the configuration domain, Felfernig proposes the way to explicitly represent the intended interpretation of these relationships in OIL. He chooses a specific ontological interpretation of the part-whole relation that is most suited for the domain at hand.

Similarities: The similarities between the mappings are shown in Table 1. In the table we present an intersection of the mappings in order to specify the set of elements which are mapped similarly in all the approaches.

4.2 Conclusions and Future Work

Comparing different approaches for transforming UML models into ontology languages for the semantic web, we recognized that there is an agreement on how to translate a subset of UML class diagrams in general (see Table 1). Beyond this common core, different approaches take very different decisions of how to translate constructs. The actual choice for a specific translation is always a result of the purpose of the translation. We think that this situation that has naturally evolved in the parallel work of different authors actually points towards a successful strategy for the transformation task. Such a strategy should start from the common core mentioned above and make more specific transformation choices based on the intended use of the resulting ontology.

Apart from this general strategy for transforming UML diagrams, the main insight we got when looking at existing approaches is the fact that there are still many open questions

OIL [9]	**DAML+OIL** [8]	**Pure UML**	**DAML** [1]
class-def c	daml:Class	class	daml:Class
class-def c	rdfs:label	class name	rdfs:label
subclass-of	rdfs:subClassOf	generalization	rdfs:subClassOf
slot-def a	daml:DatatypeProperty	attribute	daml:ObjectProperty or daml:DatatypeProperty
slot-constraint a cardinality 1 d	daml:toClass	attribute type	daml:toClass
slot-constraint	daml:ObjectProperty	binary /unidirectional association	daml:ObjectProper
domain	daml:Class rdfs:subClassOf	source class of an association	daml:Class rdfs:subClassOf
range	daml:toClass	target class of an association	daml:toClass
cardinality & **cardinality**	**multiplicity** & **cardinality**		
min-cardinality 1 max-cardinality 1	daml:cardinality=”1”	1	Unique/UnambiguousProperty daml:cardinality=”1”
absence	absence	0..*	absence
min-cardinality 1	daml:minCardinality=”1”	1..*	daml:minCardinality=”1”
min-cardinality 0 max-cardinality 1	daml:minCardinality=”0” daml:maxCardinality=”1”	0..1	daml:minCardinality=”0” daml:maxCardinality=”1” Unique/UnambiguousProperty
min-cardinality n max-cardinality n	daml:cardinality=”n”	n	daml:cardinality=”n”
min-cardinality m max-cardinality n	daml:minCardinality=”m” daml:maxCardinality=”n”	m..n	daml:minCardinality=”m” daml:maxCardinality=”n”

Table 1: Overview of similar transformation choices.

concerning the transformation between UML and OWL. We discuss open issues we consider important in the following.

Implementation The question of how to implement these transformations comes naturally, and indeed we can report distinct approaches in this sense. A very simple way to perform these transformations is to take advantage of the XML encoding used in XMI (the serialization of UML) and DAML and to write an XSLT file that defines the transformation between the two tree structures. This method was used in [5] to translate XMI to RDF(S) and java files. We used this method in transforming XMI to DAML+OIL, as presented in [8]. In spite of the simplicity of this implementation, our experience shows that XSLT is quite cumbersome to use when one wants to express more complex mappings. In addition, it is very sensitive to the format of the input file.

There exist at least two initiatives which provide plug-ins to UML editors extending their functionality with the possibility to export UML diagrams into DAML. Within the CODIP project [1] a plug-in was developed which can be used both with Rational Rose and with another UML editor, ArgoUML. Similarly, the UBOT [2] tool suite includes a module for translating UML to DAML. These tools are currently in their early development stages and we expect more effort to be required in developing robust tools for such translations.

Reasoning Existing UML editors, such as RationalRose [3], offer basic services for checking UML models for syntactic errors. The range of support of checking the correctness of a model gets considerably wider if the model is translated to a formal language.

[1]http://sylvester.va.grci.com/codipsite/codipsite/index.html
[2]http://ubot.lockheedmartin.com/ubot/
[3]http://www.rational.com/

First, one can check the consistency of the models. This aspect is especially important when it comes to ontology development and testing. Indeed, existing ontology editors offer a consistency checking service. In the UBOT project the ConsVISor [2] tool is used to check the consistency of the modelled ontologies. Based on Prolog (and planned to be migrated to Jess), this tool returns both parsing and consistency errors. The SNARK theorem prover is used to check logical inconsistencies. Similarly to UBOT, the OilEd ontology editor uses FaCT (Fast Classification of Terminologies) [13] for consistency checking. FaCT is a Description Logic (DL) classifier that can also be used for modal logic satisfiability testing.

Second, a formal representation allows deriving new knowledge by using inference engines built for that particular representation. For example FaCT can derive new facts given an original model. A direct way of assessing the quality of the existent transformation schemes would be to (1) check the quality of their output and (2) measure the amount of information that can be derived from their outputs.

Beyond Class Diagrams The mappings that we have described address the content of class-diagrams. However, as we have stated earlier in this paper, UML is much richer. The class diagrams have been in the center of attention because there exists a direct relation between their elements and the parts of an ontology (classes, hierarchies, class attributes). However, other types of diagrams express different types of information. For example state-charts or activity diagrams are useful for service and process related ontologies. In this sense a proposal exists to model DAML-S using UML diagrams [4]. There is no doubt that other UML diagrams can also be used to model various knowledge aspects of the Semantic Web.

Rules UML provides a language for expressing rules, OCL. OCL is more complex than the diagrams, however we can suppose that UML users have more experience with OCL then with DAML. Also previously developed ontologies often describe axioms using OCL. These considerations imply that the translation from OCL to DAML would be beneficiary both for supporting modelling and exploiting existing models. The research in this direction is hampered by two major problems. First, OCL does not have formal semantics and therefore it is difficult to map it into a formal language. However there are more proposals for a formal semantics for OCL [4]. Second, DAML is not developed to express rules. However there is research going on to develop a rule language for the Semantic Web (DAML-L, RuleML).

Extended support for modelling With the existent transformations it is not guaranteed that transforming a UML model to DAML and then transforming the DAML file back to UML would lead to the same diagrams. A transformation that would preserve all information would allow modelling and inferencing in parallel: a UML model would be transformed to a formal language, new information would be derived and then presented back to the modeler in UML format. This is a desirable functionality for powerful modelling tools.

Another observation is that the elements of class-diagrams are only enough to model light-weight ontologies. Enriching these ontologies with rules and axioms would be possible with OCL. Therefore, to build complex ontologies using UML the modelling tools have to support transformations both for class-diagrams and OCL rules.

[4]http://codip.grci.com/codipsite/wwwlibrary/DUETGuide/DAMLS-UML_Mapping_V1.htm

Acknowledgements

We would like to thank Lynda Hardman, Martin Gogolla and Borys Omelayenko for valuable comments for improvement and Frank van Harmelen for suggestions in early stages of the work.

References

[1] K. Baclawski, M. Kokar, P. Kogut, L. Hart, J. Smith, W. Holmes, J. Letkowski, and M. Aronson. Extending UML to Support Ontology Engineering for the Semantic Web. In *Fourth International Conference on UML*, Toronto, October 1-5 2001.

[2] K. Baclawski, M. Kokar, R. Waldinger, and P. Kogut. Consistency Checking of Semantic Web Ontologies. In *First International Semantic Web Conference (ISWC 2002)*, Sardinia, Italy, June 2002.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):35–43, 2001.

[4] M.V. Cengarle and A. Knapp. A Formal Semantics for OCL 1.4. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference Proceedings*, volume 2185 of *LNCS*, pages 118–133, Toronto, Canada, October 2001. Springer.

[5] S. Cranefield. UML and the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, 2001. http://www.semanticweb.org/SWWS/program/full/paper1.pdf.

[6] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web Ontology Language (OWL) Reference Version 1.0. Working draft, W3C, November 2002. http://www.w3.org/TR/owl-ref/.

[7] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.

[8] K. Falkovych. Ontology Extraction from UML Diagrams. Master's thesis, Vrije Universiteit Amsterdam, August 2002.

[9] A. Felfernig, G. Friedrich, and D. Jannach. UML as domain specific language for the construction of knowledge based configurations systems. *International Journal on Software Engineering and Knowledge Engineering*, 10(4):449–470, 2000.

[10] D. Fensel, I. Horrocks, F. van Harmelen, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–44, 2001.

[11] D. Fensel, F. van Harmelen, Y. Ding, M. Klein, H. Akkermans, J. Broekstra, A. Kampman, J. van der Meer, Y. Sure, R. Studer, U. Krohn, J. Davies, R. Engels, V. Iosif, A. Kiryakov, T. Lau, U. Reimer, and I. Horrocks. On-To-Knowledge in a Nutshell. *IEEE Computer*, 2002.

[12] J. Hendler and D. L. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):67–73, November/December 2000.

[13] I. Horrocks. The FaCT system. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 307–312. Springer-Verlag, Berlin, May 1998.

[14] Object Management Group. Unified Modeling Language (UML). http://www.omg.org/uml/.

[15] Object Managemet Group. Meta-Object Facility (MOF) Specification. http://cgi.omg.org/docs/formal/00-04-03.pdf, April 2002.

[16] F. van Harmelen and I. Horrocks. Reference description of the DAML+OIL ontology markup language. http://www.daml.org/2000/12/reference.html, december 2000.

[17] F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference description of the DAML+OIL (march 2001) ontology markup language. http://www.daml.org/2001/03/reference.html, march 2001.