

Extending UML to Support Ontology Engineering for the Semantic Web

Kenneth Baclawski², Mieczyslaw K. Kokar², Paul A. Kogut¹, Lewis Hart⁵,
Jeffrey Smith³, William S. Holmes III¹, Jerzy Letkowski⁴, and Michael L. Aronson¹

¹ Lockheed Martin Management and Data Systems

² Northeastern University

³ Mercury Computer

⁴ Western New England College

⁵ GRC International

Abstract. There is rapidly growing momentum for web enabled agents that reason about and dynamically integrate the appropriate knowledge and services at run-time. This trend is being led by the World Wide Web Consortium and the DARPA Agent Markup Language (DAML) program. This dynamic capability depends on the existence of explicit declarative machine-readable semantic models (ontologies). DAML is an emerging language for representing ontologies on the web. DAML was designed to support tractable reasoning. We have been developing tools for modeling ontologies in UML and generating DAML. This allows the many mature UML tools, models and expertise to be applied to knowledge representation systems, especially for visualizing complex ontologies. Our paper identifies the similarities and differences (with examples) between UML and DAML. To reconcile these differences, we propose a modest extension to the UML infrastructure for one of the most problematic differences. This is the DAML concept of property which is a first-class modeling element in DAML, while UML associations are not. For example, a DAML property can relate more than one pair of classes. Our proposal is backward-compatible with existing UML models while enhancing its viability for ontology modeling.

1 Introduction and Motivation

Representing knowledge is an important part of any knowledge-based system. In particular, all artificial intelligence systems must support some kind of knowledge representation. Because of this, many knowledge representation languages have been developed. Recently the knowledge representation community has started converging to a single knowledge representation language, called the Resource Description Framework (RDF) [11]. RDF is a recommendation within the XML suite of standards developed under the auspices of the World Wide Web Consortium. RDF is developing quickly [4]. There is now an RDF Schema language, and an extension called the DARPA Agent Markup Language (DAML) [8, 13] is now being developed that will be able to express constraints and logical inference.

The schema for a knowledge-based system is called an *ontology* [6, 7, 10]. An ontology is an explicit, formal, machine-readable semantic model. DAML is emerging as the standard for specifying ontologies. The increasing interest in ontologies is driven by many forces, but one that is rapidly gaining momentum is the emergence of web-enabled agents [9]. These agents can reason about and dynamically integrate the appropriate knowledge and services at run-time. Formal ontologies are the basis for such agents. DAML is designed to support agent communication and reasoning. We have been developing tools for constructing and checking DAML ontologies.

Neither RDF nor DAML currently has a standard graphical representation, although some informal representations have been used. We are currently engaged in a project that uses UML as the graphical representation language for DAML. Although their purposes are somewhat different, UML and DAML share many characteristics. For example, both have a notion of a *class* which can have *instances*, and the DAML

notion of *subClassOf* is essentially the same as the UML notion of specialization/generalization. Table 2 lists our best attempt to capture the similarities between the two languages.

The application of UML to the DAML initiative allows for a large number of existing UML-based tools and models as well as the considerable experience with UML to be used for knowledge representation systems. Using UML as a graphical front-end for modeling ontologies in a text-based DAML knowledge representation system would allow UML to be used for visualization of complex ontologies.

Our paper identifies the similarities and differences (with examples) between UML and DAML. To reconcile these differences, we propose a modest extension to the UML infrastructure for one of the most problematic differences. This is the DAML concept of *property*. A DAML property, at a first glance, looks the same as a UML binary association. This is misleading, since the DAML notion of property is a first-class modeling element, while UML associations are not. More precisely, in DAML, a property can exist without specifying any classes that it might relate, i.e., it can exist independently of any classes. In UML, on the other hand, an association is defined in terms of *association ends*, which must be related to classifiers. Moreover, while in UML a binary association can only relate two classes, in DAML, a property can relate more than one pair of classes.

After analyzing the differences between the two modeling languages, we came to the conclusion that it would not take too much to close the gap in the expressibility of UML while remaining backward-compatible with existing UML models. Future work will highlight other proposals that would enhance its viability for ontology modeling. Eventually, this work will culminate in a contribution to the UML and the MOF [5], in the form of a Profile or Infrastructure and MOF 2.0 recommendation, to lead to the acceptance of UML by the DAML community as the graphical notation for DAML.

2 DAML Background

The aim of the DAML program is to achieve “semantic interoperability between Web pages, databases, programs, and sensors.” An integration contractor and sixteen technology development teams are working to realize the DAML vision of “providing a set of tools for programmers to build broad concepts into their Web pages ... and allowing the bottom-up design of meaning while allowing higher-level concepts to be shared.” The problem DAML addresses is how to build a monolithic set of ontologies upon agreed-upon domain models to share in a military grid. The solution is to develop usable interoperability technologies, similar to those that enable the web to function. Towards this end, DAML will annotation information on the web to make it machine-readable so that software agents can interpret and reason with the meaning of web information.

DAML is only part of the *Semantic Web* vision [2] of the automation or enabling of things that are currently difficult to do: locating content, collating and cross-relating content, drawing conclusions from information found in two or more separate sources. DAML’s part is to serve as a markup language for network agents to provide a mechanism for advertising and reusing specifications. The software tools for creating these agents will be accomplished through the TASK (Taskable Agent Software Kit) Program to reduce the per agent development cost. The third part of the Semantic Web vision addresses the middleware layer as a continuation of the CoABS (Control of Agent Based Systems) investment to bring systems, sensors, models, etc. into the prototype “agent grid” as an infrastructure for the run-time integration of heterogeneous multi-agent and legacy systems.

The use of ontologies provides a powerful way to describe objects and their inter-relationships. The DAML language is being developed, as an extension to XML and the Resource Description Framework (RDF), to represent these ontologies and to annotate web pages and their links to ontologies. The latest release of the language (DAML+OIL) provides a rich set of constructs with which to create ontologies and to markup information so that it is machine readable and understandable. The idea is to continue this evolution of metadata (see Figure 1).

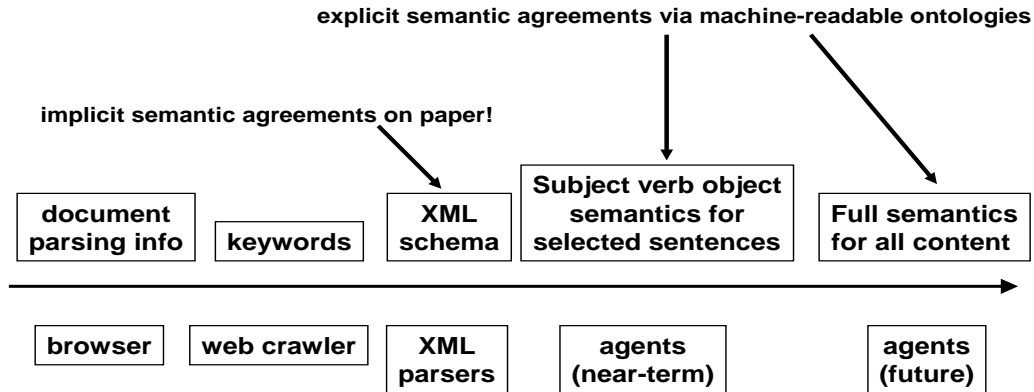


Fig. 1. The Evolution of Metadata

DAML's applications will be far-reaching, extending to both the military and commercial markets. Its machine-to-machine language capabilities might be instrumental in realizing the appliance-specific functionality, independent of human control. DAML will also enhance data-search functionality, with intelligent agents replacing inefficient search engines. Essentially, intelligent agents will be able to scan multiple Web pages and conceptually relate content that currently might seem unrelated because of variations or imprecision in the language programmers used to identify that content. A number of DAML tools have been built or are in progress, including an ontology library, parser/serializer, ontology editor/analyzer, DAML crawler and viewer, etc. Trial government (e.g. Intelink at the Center for Army Lessons Learned) and commercial (in e-commerce and information retrieval) applications have been planned and built.

The latest specification of the DAML ontology language (called DAML+OIL) was released in March, 2001. A description of the language specifications and documentation can be seen at [17]. For a good discussion of the design rationale see [12]. Also a variant called DAML-L (logic) is in progress for rule representation and reasoning.

3 Properties of Mappings

Because of the increasing number of modeling languages, it is becoming more important to introduce systematic techniques for constructing mappings (or *transformations*) between modeling languages and proving that the mappings are correct [14–16]. In this section we discuss in general terms some of the issues that arise when constructing mappings between modeling languages. When constructing mappings between modeling languages, it is important to understand the goals and purpose of the mappings. A precise statement of goals and purpose is essential for dealing with the many mapping issues, such as the following:

- Is the mapping required to preserve semantics? Ideally, the two languages should have well-defined notions of semantics. In practice, they will not, so the best one could hope for is to have some reasonably precise and convincing argument that the semantics are preserved.
- Is the mapping required to be defined on the entire modeling language? In many cases, it may suffice to define the mapping on a subset of the modeling language. The purpose of the mapping can be used to answer this question. If the language is simply a means (or “front-end”) for constructing models the second language, then it is reasonable to use only those constructs of the first language that are needed for the second.

- Is the mapping simply a one-way mapping from one language to the other or should it be defined in both directions? If the mapping is defined in both directions, then it is called a *two-way* mapping.
- If the mapping is a two-way mapping, should the two directions be inverses of each other? Having inverse mappings is generally only possible when the languages are very similar to one another. This is not the case for UML and DAML.

To make the discussion of mapping properties more precise, we need to introduce some concepts. We presume that each modeling language has notion of *semantic equivalence*. The precise meaning of this notion will depend on the language, but it usually takes a form such as the following: Two models M_1 and M_2 are *semantically equivalent* if there is a one-to-one correspondence between the instances of M_1 and the instances of M_2 that preserves relationships between instances. Semantic equivalence of two models should mean that the models differ from each other only in inessential ways, such as renaming, reordering or adding redundancy.

We also presume that each model of a language can be serialized in a unique way. For example, one can serialize a UML model using the XMI format, while DAML is defined in terms of RDF which has a standard XML representation. For a model M in a language L , the *size* of M is the size of its serialization (in whatever unit is appropriate for the serialization, such as the number of characters). The size of M is written $\#M$.

Now suppose that L_1 and L_2 are two modeling languages. A *mapping* f from L_1 to L_2 is a function from the models of L_1 to the models of L_2 which preserves semantic equivalence. In other words, if M_1 and M_2 are two semantically equivalent models in L_1 , then $f(M_1)$ is semantically equivalent to $f(M_2)$. In the case of UML and DAML, the mapping is defined only on those UML models that are necessary for expressing DAML ontologies, so it is only a partial mapping. The subset on which the mapping is defined is called DUML.

A *two-way mapping* from L_1 to L_2 is a pair of mappings, the first f_1 from L_1 to L_2 and the second f_2 from L_2 to L_1 , such that if f_1 is defined on M , then f_2 is defined on $f_1(M)$, and vice versa for f_2 and f_1 . By assumption, two-way mappings preserve semantics in both directions

In general, two-way mappings are not inverses, even for the models on which they are defined. The best one can hope for is that applying the two mappings successively will stabilize, but even this is hard to achieve. To be more precise, we say that a two-way mapping is *stable* if for any model M on which f_1 is defined, $f_1(f_2(f_1(M))) = f_1(M)$, and similarly for models of L_2 . While stability is much easier to achieve than invertibility, it is still a strong property of mappings. Let (f_1, f_2) be a stable two-way mapping. For any model M on which f_1 is defined, $f_2(f_1(M))$ forms a kind of “canonical form” for M in the sense that f_1 and f_2 are inverses of each other on the canonical forms. Put another way, stable two-way mappings furnish canonicalizations for the two languages as well as invertible mappings between canonical forms.

While stability is clearly desirable, it may not be necessary. A more realistic goal is for the two-way mapping to settle down eventually. To be more precise, a two-way mapping (f_1, f_2) is *bounded* if for any model M on which f_1 is defined, the sequence $\#f_1(M), \#f_2(f_1(M)), \#f_1(f_2(f_1(M))), \dots$ is bounded.

Assuring boundedness might seem easy, but even this property takes some effort to achieve. Consider, for example, a mapping from UML that maps each association to a DAML class and each association end to a DAML property. This is certainly necessary for association classes and nonbinary associations. Using the same mapping uniformly for all associations is certainly simpler than treating binary associations that are not association classes in a different manner. However, doing so is unbounded. A binary association will map to a class and two properties, which map back to a class and two associations, these then map to three classes and four properties, and so on.

4 UML to DAML Mapping

In order to discuss the similarities between UML and DAML an initial incomplete mapping between the languages has been created. Table 1 presents a high-level mapping of concepts from UML and DAML, and serves as an overview of the overall strategy applied to the mapping.

Table 2 elaborates on the high-level concepts and expresses some of the specific extensions necessary for the initial mapping between the languages. This proposed mapping is made with the assumption that UML

DAML Concept	Similar UML Concepts
Ontology	Package
Class	Class
As Sets (disjoint, union)	Difficult to represent
Hierarchy	Class Generalization Relations
Property	Aspects of Attributes, Associations and Classes
Hierarchy	None for Attributes, limited Generalization for Associations, Class Generalization Relations
Restriction	Constrains Association Ends, including multiplicity and roles. Implicitly as class containing the attribute
Data Types	Data Types
Instances and Values	Object Instances and Attribute Values

Table 1. High-Level Mapping of UML and DAML Concepts

class diagrams are created specifically for the purpose of designing DAML ontologies. Legacy class diagrams that were not originally intended for DAML applications would need modification in order to be used for DAML purposes.

4.1 Representing DAML Properties

Individual elements of this mapping can be illustrated to further explain the principles used to create the mapping. Figure 2 depicts the “mother” relationship that exists between the class Person and the class Woman. In UML this is represented as a labeled association between the two classes. In DAML the property “mother” exists independently of the two classes and is not given significance until a restriction is placed on the source class of the relationship. In Figure 2, this is represented as a restriction for class Person, on property “mother”, to class Woman.

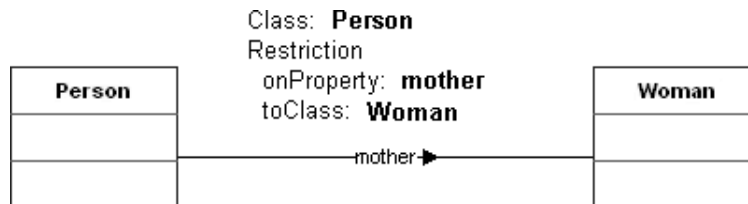


Fig. 2. DAML Property Restriction

By applying the proposed DAML to UML mapping, a DAML translation can be generated. Listing 3 represents a section of an ontology that has been constructed from Figure 2

Another concept of the mapping can be seen in Figure 4, which shows one of the UML representations of a DAML Sub-Property. In the figure, the property that represents a person as being the “father of” another person is a refinement of the property of a person being the “parent of” another person.

4.2 Representing DAML Instances

Figure 5 illustrates the concept of an instantiated class in UML. In a similar fashion, this would be described in DAML as an element identified as “Tommy”, with type identified as Person and the value “9” assigned to the property “age”.

UML	DAML
class	Class
instanceOf	type
type of ModelElement	type
attribute	ObjectProperty or DatatypeProperty
binary association	ObjectProperty
generalization	subClassOf
«subPropertyOf» stereotyped dependency between 2 associations	subPropertyOf
generalization between stereotyped classes	subPropertyOf
note	comment
name	label
“seeAlso” tagged value on a class and association	seeAlso
“isDefinedBy” tagged value on a class and association	isDefinedBy
class containing the attribute	“subClassOf” a property restriction
source class of an association	“subClassOf” a property restriction
attribute type	“toClass” on a property restriction
target class of an association	“toClass” on a property restriction
«equivalentTo» stereotyped dependency	equivalentTo
«sameClassAs» stereotyped dependency between two classes	sameClassAs
«samePropertyAs» stereotyped dependency between two associations	samePropertyAs
«Ontology» stereotyped package	Ontology
“versionInfo” tagged value on a package	versionInfo
import (dependency stereotype)	imports
multiplicity	cardinality
multiplicity range Y..Z	Y = minCardinality, Z = maxCardinality
association target with end multiplicity = 0..1 or 1	UniqueProperty
association source with end multiplicity = 0..1 or 1	UnambiguousProperty
«inverseOf» stereotyped dependency between two associations	inverseOf
«TransitiveProperty» stereotype on an association	TransitiveProperty

Table 2. Mapping Between UML and DAML

```

<daml:Class rdf:ID="Person">
  <daml:label>Person</daml:label>
  <daml:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#mother"/>
      <daml:toClass rdf:resource="#Woman"/>
    </daml:Restriction>
  </daml:subClassOf>
</daml:Class>
<daml:Class rdf:ID="Woman">
  <daml:label>Woman</daml:label>
</daml:Class>
<daml:Property rdf:ID="mother"/>

```

Fig. 3. DAML Translation of Figure 2

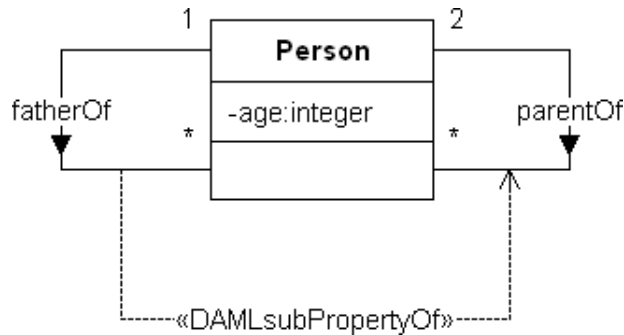


Fig. 4. Example of a Sub-Property

4.3 Representing Facets of Properties

To demonstrate the mapping between UML multiplicity and DAML cardinality, Figure 6 depicts the correlation between the multiplicity of an association end and the corresponding cardinality in DAML. In the figure, an association end that contains a single value would map to a specific cardinality value for the property restriction. An association end that contains a range of values would map to the minimum and maximum cardinality allowed for the corresponding property restriction.

Figure 7 and Figure 8 depict special cases of DAML properties with predefined cardinality restrictions. The first of these is called an Unambiguous Property and is depicted in Figure 7. An Unambiguous Property is defined in DAML as a relation that, given a specified target element, will always originate from the same source element.

Figure 8 represents the UML notation for the DAML concept of a Transitive Property. A Transitive Property is defined in the terms of three or more elements. To be considered transitive, a property that holds true for the first and second elements and holds true for the second and third elements must also hold true for the first and third elements. For example, given that Tom is the ancestor of Jack, and Jack is the ancestor of Robert, then Tom is also the ancestor of Robert.

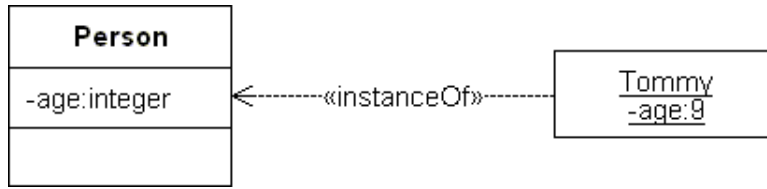


Fig. 5. DAML type property

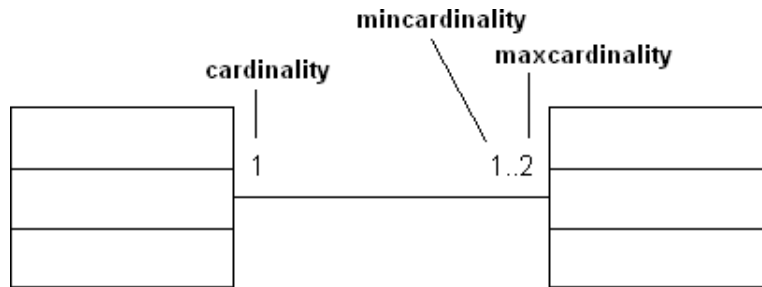


Fig. 6. DAML Cardinality

5 Incompatibilities Between UML and DAML

Using UML as a front-end to a DAML knowledge representation system would allow the large number of existing UML-based tools and considerable experience with UML to be used for knowledge-based systems. Unfortunately, while there are many similarities between UML and DAML, there are also many differences. Reconciling these differences has been one of the major problems of our project. We now discuss the major incompatibilities between UML and DAML.

5.1 Containers and Lists

RDF has a number of container notions: *Bag*, *Seq* and *Alt*. The semantics of these notions are not very clear, and DAML has largely replaced them with the notion of a *List*. Although most uses of lists can be mapped to UML ordered associations, the correspondence is not perfect.

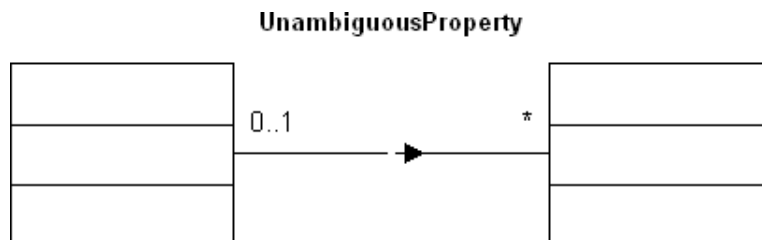


Fig. 7. Example of an Unambiguous Property

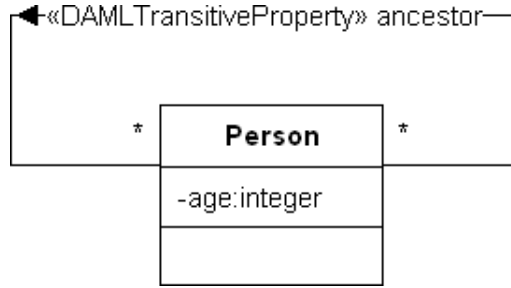


Fig. 8. Example of a Transitive Property

5.2 Universal Classes

RDFS and DAML have “universal” classes. The *Resource* class has every object as an instance, and the *Literal* has every literal as an instance. DAML adds the *Thing* class that has every object, including both objects and literals (and presumably anything else as well). No such universal classes exist in UML. One can certainly add such classes to a UML model, but it is not compatible with the spirit of UML modeling.

5.3 Constraints

DAML imposes a constraint by specifying that a class is a subclass of a restriction class. The semantics of each kind of restriction constraint is specified by the DAML axioms. UML can specify constraints using a variety of graphical mechanisms, and one can also specify constraints using OCL. The graphical constraint mechanisms (e.g., multiplicity constraints) are specified by OCL, so ultimately all constraints get imposed, directly or indirectly, in OCL.

5.4 Classes

In RDFS and DAML, a class can be an instance of another class. However, this is not commonly done. UML imposes a separation between the object and class levels, although there are situations in which one can relate objects on different meta-levels.

5.5 Property

A DAML property, at a first glance, looks the same as a UML binary association. But this is misleading, since the DAML notion of property is a first-class modeling element, while UML associations are not. More precisely, in DAML a property can exist without specifying any classes that it might relate, i.e., it can exist independently of any classes. In UML, on the other hand, an association is defined in terms of *association ends*, which in turn must be associated with classifiers. Moreover, while in UML a binary association can only relate two classes, in DAML a property can relate more than one pair of classes.

Another significant difference between UML and DAML is that the relation between UML classes and associations is not exactly the same as between DAML classes and properties. In UML, multiplicity constraints on associations can affect membership of objects in the classes related through the association; this is because multiplicity constraints constrain the number of objects that can be instantiated for these classes. Classes in UML do not directly affect associations. In DAML, constraints on properties are imposed somewhat indirectly by specifying that a class is a subtype of a class called a *restriction*. Doing this may limit the scope of the properties being constrained by the restriction class.

Finally, another important difference between UML and DAML is that descriptions of both classes and properties in DAML can be distributed over various locations. This is not in the spirit of UML.

The differences identified above have their own advantages and disadvantages. The idea of distribution of descriptions, for instance, goes against the principle of modularization, an accepted principle in software engineering, but it does help to support reuse. Similarly, the idea of a property being associated with multiple classes is more flexible and might foster reuse, but it clashes with modularity. Consider, for example, the notion of a `location`. This is a property that occurs frequently in models, often several times within the same model. In UML, such occurrences are different associations, while in DAML, they would all be the same property.

For instance, the property of `location` could associate `Faculty` with `University`. Each link of this association would give the University affiliation of a faculty member. In UML it would be modeled as an association. The same property might also be used for associating a `Building` with its `Address`. In UML, this would be modeled as a second association, whether or not the associations use the same name, because the associations are in different namespaces.

In RDF and DAML (as well as many other knowledge representation languages), properties are first class. A property need not have any domain or range specifications at all, but when it does it may have multiple domains and only one range. Furthermore, properties may have values that are literals as well as objects, so that properties subsume both the association and the attribute concepts in UML.

On the other hand, UML allows associations that are nonbinary, while properties can only be binary. There are well-known techniques for dealing with nonbinary relationships, but it is much harder to deal with the fact that UML associations and attributes cannot be first class.

Although two UML associations may have the same name, they are still different associations because the names are in different namespaces. If one chooses to map two UML associations having the same name to the same RDFS property, then this could violate the requirement that an RDFS property have only one range. If one maps each association to a different RDFS property, then RDFS properties having multiple domains will not be expressible in UML.

To deal with the problem of first class properties, we recommend that a new type of model element be added to UML for representing properties. Since RDF properties are unidirectional, it would be incorrect to view a property as a grouping of associations. The correct interpretation is to define a property to be an aggregation of associations ends from different associations. This is discussed in more detail in Section 6.

5.6 Cardinality Constraints

UML multiplicity constraints on an association end correspond relatively accurately to the DAML cardinality constraints. The only incompatibility has to do with the fact that properties are first class model elements and that properties are one-directional. The first class feature of properties means that one can specify a cardinality constraint for every domain of a property all at once. In UML one must specify this separately for each association (end) belonging to the property, while in DAML it is only necessary to specify it once. On the other hand, UML allows one to specify cardinality constraints on all of the ends of an association. In DAML, one must introduce an inverse property in order to specify a cardinality constraint on the domain of a property. Our recommendation would handle all of these issues in a relatively clean manner.

5.7 Transitivity

DAML has the capability of imposing constraints on properties. From the point of view of UML, this means that one can impose a constraint on a number of associations all at once. This is a useful modularization technique related to current work on aspect-oriented programming. The only constraint of this kind that is explicitly supported at the moment is transitivity, but other constraints may be added later.

5.8 Subproperties

RDF allows one property to be a subproperty of another. UML has the ability to specify that one association is a specialization of another, though this construct is rarely used. In our recommendation, *Property* is a

specialization of *Classifier*, so that a property can be a specialization of another. Of course, OCL constraints must be added to ensure that one does not have meaningless specializations, such as properties that are specializations of associations, and the semantics of property specialization must be specified carefully.

5.9 Namespaces

While it is reasonable to define a mapping from UML to DAML by specifying how each construct is to be mapped, one must also consider how the constructs are related to one another. In other words, in addition to the major constructs one must also consider the “glue” that ties them together.

Constructs in DAML are linked together either through the use of URIs or by using the hierarchical containment relationship of XML. DAML objects need not be explicitly named (i.e., they can be anonymous), and such objects can be related to other objects using XML containment.

UML uses a very different kind of “glue” to link its objects to each other. Instead of URIs, it uses names in a large number of namespaces. For example, each class has its own namespace for its attributes and associations. RDF also has namespaces (from XML), but the XML namespaces are a very different notion.

Any mapping from UML to DAML or the reverse must have a mechanism for ensuring that names are properly distinguished. However, there are known methods for dealing with this problem, and no new UML features are needed to deal with this.

UML also uses graphical proximity to specify relationships, and these are the only way that unnamed objects can be linked with other objects. Graphical relationships are more complex than hierarchical containment, and one would expect that graphical interfaces would be more general than hierarchical containment. However, this is not quite true. The XML serialization form of RDF can specify sequence information very easily while it is awkward to specify a sequential order for graphical objects. Indeed, serializations impose a sequence ordering on every relationship even when it is irrelevant.

When specifying a mapping from UML to DAML, one should also address the issue of how relationships between model elements are to be mapped. The most important issue is the mapping of names, but other issues are also significant.

6 Recommendations

DAML is similar to many other knowledge representation languages that are based on the mathematical notion of a graph or network (consisting of a set of vertices and edges). Conceptual graphs and semantic networks are examples of commonly used knowledge representation languages of this kind. Natural Language Processing (NLP) systems are well suited to this kind of knowledge representation because an edge from one vertex to another corresponds to a predicate linking a subject to an object. Parts of speech in general map reasonably well to modeling constructs in knowledge representation systems (see, for example, [1]). In DAML a predicate is represented by a property. However, the DAML notion of a property is defined independently of any context in which it might be used. Whether properties should be *decontextualized* in this manner is a hotly debated philosophical issue.

We do not take any particular stand on whether decontextualized properties are appropriate for modeling activities. Rather we feel that this decision should be left to the modeler. Furthermore, the knowledge representation community is a large and growing community, and it makes sense to support their modeling techniques if it is convenient to do so and it does not break any existing models. We argue here that by adding a few additional model elements to the UML metamodel one can make UML compatible with knowledge representation languages.

After analyzing the differences between the two modeling languages we came to the conclusion that it would not take too much to close this gap in the expressibility of UML. Towards this end, we propose to extend UML by adding two meta-model elements called *Property* and *Restriction*. The MOF diagram for this extension is shown in Figure 9. As can be seen from the diagram, *Property* is an aggregation of a number of association ends. The notion of Association End does not need to be changed. The notion of *Property* serves as a means of grouping various association ends. This capability is not present in the current UML.

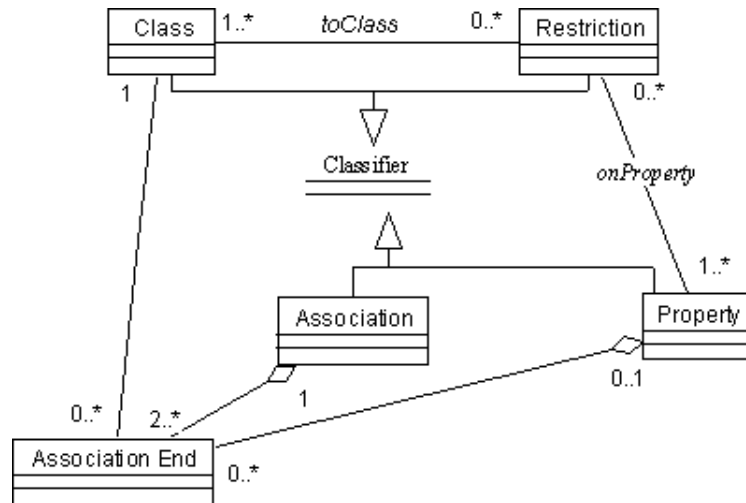


Fig. 9. The MOF Specification of the UML Extension

The fact that *Property* is a first-class concept is shown by the fact that *Property* can exist without being associated with any classes. This is imposed by setting the multiplicity constraint on the aggregation to $0..*$. A property can be constrained by zero or more *Restrictions*, as is the case in DAML. The *Restriction* is a Classifier. It is also related to at least one class.

6.1 Property Semantics

A property is a grouping of association ends. Properties “cross-cut” the Association concept. In particular, no property can have more than one of the association ends of an association. To express this in OCL one uses `allConnections`, the set of all Association Ends of an Association, and we introduce `allPropConnections` to be the set of all Property classifier of an Association. If T is the intersection of the `allConnections` and `allPropConnections` sets, then T has cardinality at most 1. More formally:

```

allConnections: Set(AssociationEnd);
allPropConnections: Set(Property);
self.allConnections->intersection(self.allPropConnections:Set(T)):Set(T);
size(#T) <= 1
  
```

In addition, one must specify that Property classifier can only be specializations or generalizations of other Property classifier.

6.2 Restriction Semantics

A restriction is a classifier for objects. The instances of the restriction are the objects that satisfy a condition on one or more properties associated with the restriction. A restriction is imposed on a class by specifying that the class is a specialization of the Restriction classifier.

If a Restriction classifier is linked with a Property classifier, and if the Restriction classifier is linked with Classes (via the `toClass` meta-Association), then the instances of the Restriction classifier can only link with objects that are in one of the specified classes.

As with the Property classifiers, the Restriction classifiers can only be generalizations and specializations of other Restriction classifiers.

7 Conclusion

In this paper we have reported on our work in progress on a UML-based front-end for DAML. We have identified similarities and differences between UML and DAML. In the “similarities” discussion we showed how UML concepts can be mapped to DAML. In the “incompatibilities” discussion we identified differences between the two representations. In the “mapping” discussion, we made an attempt to give rules for translating UML concepts to DAML concepts. As a result of our analysis, we came to the conclusion that some of the concepts are significantly incompatible. In particular, the concept of DAML **Property**, although somewhat similar to the UML **Association** concept, cannot be mapped easily. We believe that this is the main obstacle to using UML (and UML tools) as a front-end to the ontology building tools that use DAML as their ontology representation language. We believe this obstacle could be reconciled by a modest extension to the UML. We proposed the main idea of such an extension in this paper. We also explained the advantages and disadvantages of having the concept of Association and Property. If the extension as proposed in this paper is accepted, then the two concepts can be mapped consistently. This might lead to the acceptance of UML by the knowledge representation community as the preferred graphical notation for knowledge representation languages, such as DAML, that are based on graphs.

References

1. R. Abbott. Program design by informal english descriptions. *Comm. ACM*, 26(11), 1983.
2. Semantic Web Activity. www.w3.org/2001/sw, 2001.
3. G. Booch, I. Jacobsen, and J. Rumbaugh. *OMG Unified Modeling Language Specification*, March 2000. Available at www.omg.org/technology/documents/formal/unified_modeling_language.htm.
4. S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL'98 - The Query Language Workshop*. W3C, 1998.
5. Object Management Group. *Meta Object Facility (MOF) Specification, Version 1.3*, April 2000. Available at www.omg.org/technology/documents/formal/meta.htm.
6. J. Heflin, J. Hendler, and S. Luke. Coping with changing ontologies in a distributed environment. In *AAAI-99 Workshop on Ontology Management*. MIT Press, 1999.
7. J. Heflin, J. Hendler, and S. Luke. SHOE: A knowledge representation language for Internet applications. Technical Report www.cs.umd.edu/projects/plus/SHOE, Institute for Advanced Studies, University of Maryland, 2000.
8. J. Hendler and D. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15, No. 6:67–73, 2000.
9. D. McGuinness. Ontologies and online commerce. *IEEE Intelligent Systems*, Vol. 16, No. 1:8–14, 2001.
10. D. L. McGuinness, R. Fikes, J. Rice, and S. Wilde. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, 2000.
11. Resource Description Framework (RDF) Model and Syntax Specification. www.w3.org/tr/rec-rdf-syntax, February 1999.
12. DAML+OIL Design Rationale. www.cs.man.ac.uk/horrocks/Slides/index.html, 2001.
13. DARPA Agent Markup Language Web Site. www.daml.org, 2001.
14. J. Smith. *UML Formalization and Transformation*. PhD thesis, Northeastern University, Boston, MA, December 1999.
15. J. Smith, M. Kokar, and K. Baclawski. Formal verification of UML diagrams: A first step towards code generation. In *Eighth OOPSLA Workshop on Behavioral Semantics*, pages 206–220, November 1999.
16. J. Smith, M. Kokar, K. Baclawski, and S. DeLoach. UML formalization and transformation. In *ECOOP*, 2000.
17. DAML+OIL Specification. www.daml.org/2001/03/daml+oil-index.html, 2001.