Extending UML to Support Ontology Engineering for the Semantic Web

Kenneth Baclawski[2], Mieczyslaw K. Kokar[2], Paul A. Kogut[1], Lewis Hart[5], Jeffrey Smith[3], William S. Holmes III[1], Jerzy Letkowski[4], and Michael L. Aronson[1]

[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. The World Wide Web Consortium and the DARPA Agent Markup Language (DAML) program have been actively involved in furthering this trend. The dynamic integration of knowledge and services depends on the existence of explicit declarative semantic models (ontologies). DAML is an emerging language for specifying machine-readable ontologies on the web. DAML was designed to support tractable reasoning.

We have been developing tools for developing ontologies in the Unified Modeling Language (UML) and generating DAML. This allows the many mature UML tools, models and expertise to be applied to knowledge representation systems, not only for visualizing complex ontologies but also for managing the ontology development process. Furthermore, UML has many features, such as profiles, global modularity and extension mechanisms that have yet to be considered in DAML.

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 have more than one domain class. Our proposal is backward-compatible with existing UML models while enhancing its viability for ontology modeling.

While we have focused on DAML in our research and development activities, the same issues apply to many of the knowledge representation languages. This is especially the case for semantic network and concept graph approaches to knowledge representations.

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 (KR). Because of this, many KR languages have been developed. For an excellent introduction to knowledge representations and ontologies see [20].

Expressing knowledge in machine-readable form requires that it be represented as data. Therefore it is not surprising that KR languages and data languages have much in common, and both kinds of language have borrowed ideas and concepts from each other. Inheritance in object-oriented programming and data languages was derived to a large extent from the corresponding notion in KR languages.

KR languages can be given a rough classification into three categories:

- Logical languages. These languages express knowledge as logical statements. One of the best-known examples of such a KR language is the Knowledge Interchange Format (KIF) [6].
- Frame-based languages. These languages are similar to object-oriented database languages.

– Graph-based languages. These include semantic networks and conceptual graphs. Knowledge is represented using nodes and links between the nodes. Sowa's conceptual graph language is a good example of this [20].

Unlike most data modeling languages, KR languages do not have a rigid separation between meta-levels. While one normally does maintain such a separation to aid in understanding, the languages do not force one to do so. In effect, all of the statements in the languages are in a single space of statements, including relationships such as "instanceOf" that go between metalevels. In DAML, as in many other KR languages, `Class` is an entity whose instances are classes. A class can have instances, and those instances may also be classes. A chain of "instanceOf" links may be of any length. The `Class` entity, in particular, is an instance of itself. As a result, KR languages incorporate not only modeling capabilities, but at the same time they include meta-modeling, meta-meta-modeling, etc.

On the other hand, DAML and most other KR languages do not have profiles, packages or other modularity mechanisms. DAML does make use of XML namespaces, but only for disambiguating names, not as a package mechanism.

The analogy between hypertext and semantic networks is compelling. If one identifies semantic network nodes with Web resources (specified by Universal Resource Identifiers or URIs) and semantic network links with hypertext links, then the result forms a basis for expressing knowledge representations that could span the entire World Wide Web. This is the essence of the Resource Description Framework (RDF) [14]. RDF is a recommendation within the XML suite of standards, developed under the auspices of the World Wide Web Consortium. RDF is developing quickly [5]. There is now an RDF Schema language, and there are many tools and products that can process RDF and RDF Schema. The DARPA Agent Markup Language (DAML) [16, 10] is an extension of RDF and RDF Schema that will be able to express a much richer variety of constraints as well as support logical inference.

As in any data language, KR languages have the ability to express schemas that define the structure and constraints of data (instances or objects) conforming to the schema. A schema in a KR language is called an *ontology* [8, 9, 12]. An ontology is an explicit, formal semantic model. Data conforming to an ontology is often referred to as an *annotation*, since it typically abstracts or annotates some natural language text (or more generally a hypertext document). An ontology may include vocabulary terms, taxonomies, relations, rules/assertions. An ontology should not include instances/annotations.

The increasing interest in ontologies is driven by the large volumes of information now available as well as by the increasing complexity and diversity of this information. These trends have also increased interest in automating many activities that were traditionally performed manually. Web-enabled agents represent one technology for addressing this need [11]. These agents can reason about knowledge and can dynamically integrate 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 developing and testing DAML ontologies and knowledge representations.

RDF and DAML, which currently do not have any standard graphical form, could leverage the UML graphical representation. In addition, RDF and DAML are relatively recent languages, so there is not as many tools or as much experience as there is for UML. We are currently engaged in projects that have realized benefits in productivity and clarity by utilizing UML class diagrams to develop and to display complex DAML ontologies. Cranefield [4] has also been promoting ontology development using UML and has been translating UML to RDF. Although their purposes are different, UML and DAML have many characteristics in common. 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.

Our paper discusses the similarities and differences between UML and DAML and they might be reconciled. We are proposing a modest extension to the UML infrastructure to deal with the DAML concept of *property*, which represents one of the most problematic differences. The DAML concept of *property* was recently split into the notions of *ObjectProperty* and *DatatypeProperty*. A DAML ObjectProperty, at a first glance, appears to be the same as a UML association, and a DAML DatatypeProperty appears to be the same as a UML attribute. This is misleading, since the DAML notion of ObjectProperty is a first-class

modeling element, while UML associations are not. More precisely, in DAML, an ObjectProperty 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. Similar remarks apply to DAML DatatypeProperties versus UML attributes. This difference between UML and most other KR languages has also been noted by Cranefield [4].

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 UML and MOF [7], in the form of a Profile or Infrastructure and UML 2.0/MOF 2.0 recommendation, to lead to more general acceptance of UML as a development environment for ontologies based on DAML and other KR languages. The recommended changes to UML would have some effect on existing tools, but the changes are not any more significant than other changes that being considered for UML 2.0 which would also have an impact on existing tools.

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 enable annotating information on the web to make knowledge about the document machine-readable so that software agents can interpret and reason with the meaning of web information. The only mechanism currently generally available for such annotations on the Web is metadata in the head element of an HTML file. DAML enriches and formalizes metadata annotations (see Figure 1).

DAML is only part of the *Semantic Web* vision [2, 13] 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.

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 application-specific functionality, independent of human control. DAML will also enhance the efficiency and selectivity of search engines and other automated document processing tools. Such engines and tools 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 been seen at [21]. For a good discussion of the design rationale see [15]. Also a variant called DAML-L (logic) is in progress for rule representation and reasoning. DAML+OIL is the basic representation language (analogous to the UML basic diagrams),

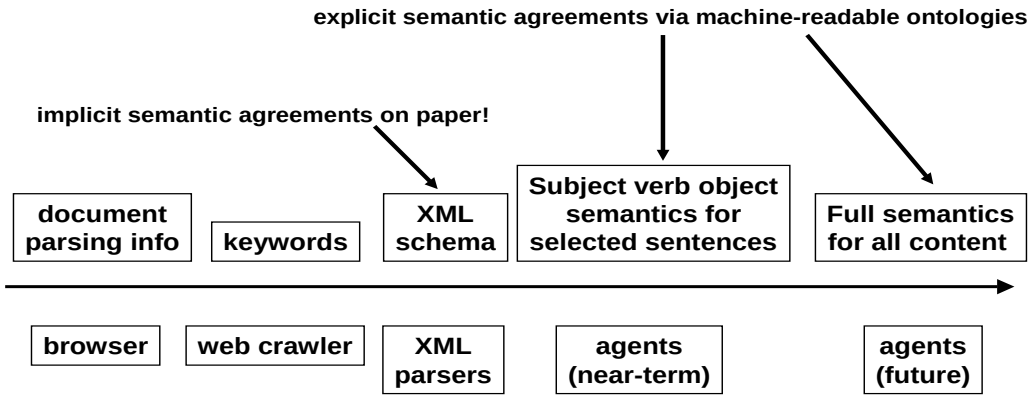

Fig. 1. The Evolution of Metadata

while DAML-L will provide for logical assertion (analogous to the Object Constraint Language (OCL) of UML).

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 [17–19]. 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 of 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.

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 $\#f_1(M), \#f_2(f_1(M)), \#f_1(f_2(f_1(M))), \ldots$ is bounded.

While it is desirable for mappings to be bounded, this can conflict with the desire to keep the mapping simple. 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. This example illustrates how keeping the mapping simple can result in unbounded mappings. We intend to propose mappings that are bounded, even though this may make them somewhat more complex.

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

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

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

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

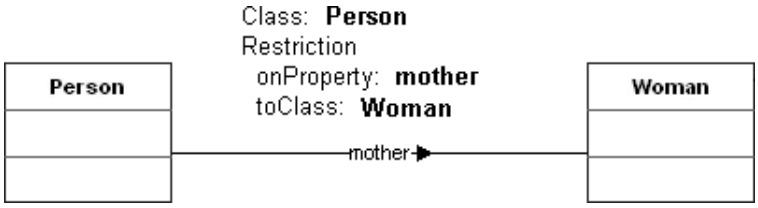

Fig. 2. DAML Property Restriction

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.

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

```
<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

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 described in DAML as an element identified as "Tommy", with type identified as Person and the value "9" assigned to the property "age".

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.

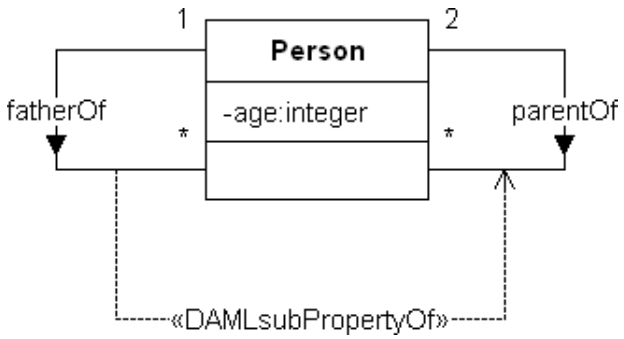

Fig. 4. Example of a Sub-Property

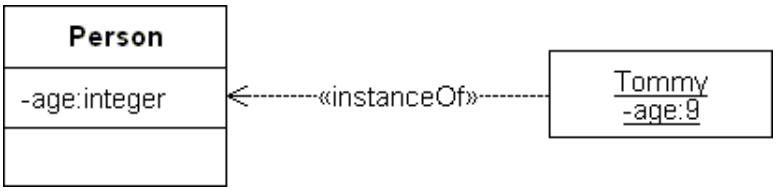

Fig. 5. DAML type property

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.

5 Incompatibilities Between UML and DAML

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*. UML does have containers (in OCL), and it also has ordered associations which implicitly define a list within the context of the association. For a particular modeling task, one can often use either one (e.g., a design using an explicit list structure could be redesigned to use an ordered association instead). However, lists and ordered associations have different interfaces, and it is difficult to map one to the other in an automated fashion.

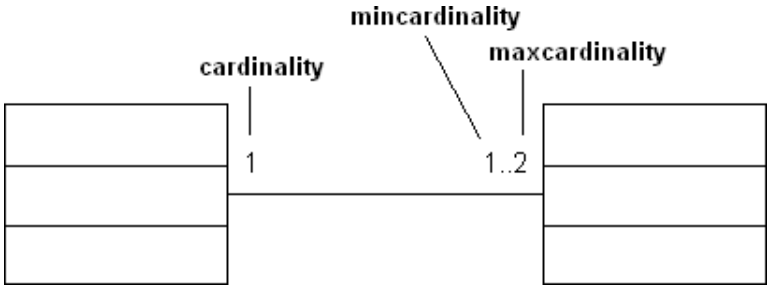

Fig. 6. DAML Cardinality

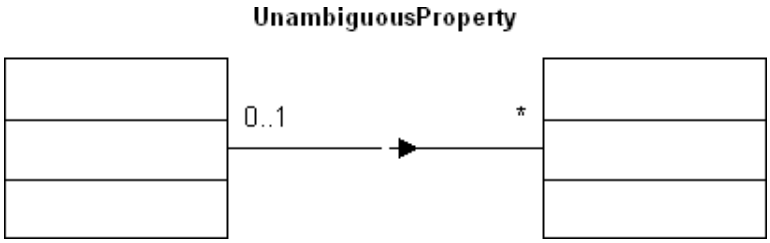

Fig. 7. Example of an Unambiguous Property

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 Property

As we have noted in Section 1 above, the DAML notion of property is a first-class modeling element, while the UML notion of an association is not. Furthermore, a UML binary association always has just one domain class and one range class, but a DAML property can have many domain classes, although it can only have one range class.

A UML nonbinary association cannot be directly represented as a single DAML property, and it must be reified as a DAML class with as many properties as the arity of the UML association. In other words, a UML nonbinary association must be *reified*. Of course, binary associations can also be reified. Whether to reify an

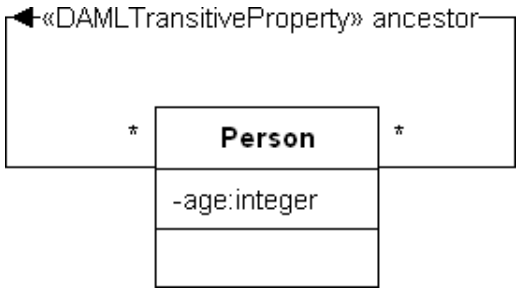

Fig. 8. Example of a Transitive Property

association is a design choice. Ontology developers refer to such design choices as *ontological commitment*. Reification is a useful design technique, but it has the disadvantage that the design is more complex, and the design units that make up the reification are no longer explicitly related to one another. In addition, if one uses reification in an automated mapping from one language to another (e.g., from DAML to UML), then the resulting mapping is unbounded, as the example at the end of Section 3 illustrates.

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 7.

5.5 Cardinality Constraints

UML multiplicity constraints on an association end correspond relatively accurately to the UML 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.

5.6 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.

5.7 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.

5.8 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. This is further enriched by the ability to specify private, protected and public scopes. RDF also has namespaces (from XML), but XML namespaces are a very different notion. RDF lacks any kind of name scoping mechanism. In addition, one cannot specify navigability constraints for RDF properties. While RDF properties are unidirectional, this is only a mechanism for distinguishing the roles of objects being related. It does not limit accessibility.

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 Semantics of Constraints

One overriding distinction between UML and DAML is the semantics of constraints. In UML a constraint is a requirement that restricts the instantiations that are possible for a given model. In DAML and other logic-based KR languages, constraints are axioms from which one can perform logical inference. To understand the distinction, suppose that in a UML class diagram one has `Student` and `Department` classes, and one has an association `major` that specifies the department in which a student is majoring. Assume that there is a cardinality constraint on `major` that constrains a student to major in at most one department. Now suppose that a particular student is majoring both in `Computer Science` and `Chemistry`. In UML this would violate the cardinality constraint. In DAML, on the other hand, one can conclude that `Computer Science` and `Chemistry` must be the same department.

7 Recommendations

DAML is similar to many other KR 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 KR 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 KR 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.

To close the gap in the expressibility of UML, 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. 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.

It is tempting to deal with the issue of first-class properties by simply reifying them. Classes are first-class modeling elements, so this appears to solve the problem. For example, instead of attempting to model `location` as an association, one could model it as a class `Location`. However, this has several disadvantages.

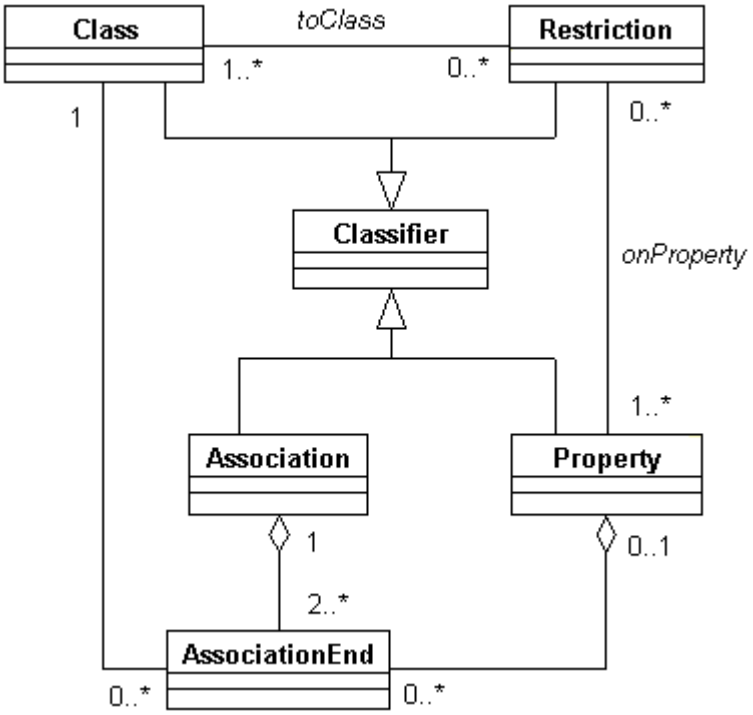

Fig. 9. The MOF Specification of the UML Extension

It can result in complex, unnatural ontologies, and it puts the burden on the ontology developer to deal with this incompatibility issue. Furthermore, if this is used as a mechanism for mapping between DAML and UML, then the resulting mapping is unbounded, as has been discussed in Section 5.4.

7.1 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.

7.2 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.

8 Conclusion

In this paper we have reported on our work in progress on a UML as an ontology development environment. We have identified similarities and differences between UML and DAML, and we have discussed how they can be mapped to each other. 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) for DAML-based ontology development. 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 KR languages, such as DAML, that are based on graphs.

Acknowledgements

This material is based upon work supported by the Air Force Research Laboratory, Contract Number F30602-00-C-0188.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Unites States Air Force.

References

1. R. Abbott. Program design by informal english descriptions. *Comm. ACM*, 26(11), 1983.
2. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 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. Cranefield. Networked knowledge representation and exchange using uml and rdf. *J. of Digital information*, 1(8), February 2001.
5. 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.
6. M. Genesereth. Knowledge interchange format draft proposed american national standard (dpans) ncits.t2/98-004, 1998. Available at `logic.stanford.edu/kif/dpans.html`.
7. Object Management Group. *Meta Object Facility (MOF) Specification, Version 1.3*, April 2000. Available at `www.omg.org/technology/documents/formal/meta.htm`.
8. 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.
9. 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.
10. J. Hendler and D. McGuinness. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15, No. 6:67–73, 2000.
11. D. McGuinness. Ontologies and online commerce. *IEEE Intelligent Systems*, Vol. 16, No. 1:8–14, 2001.

12. 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.
13. E. Miller, R. Swick, D. Brickley, and B. McBride. Semantic web activity page, April 2001. Available at `www.w3.org/2001/sw/`.
14. Resource Description Framework (RDF) Model and Syntax Specification. `www.w3.org/tr/rec-rdf-syntax`, Feburary 1999.
15. DAML+OIL Design Rationale. `www.cs.man.ac.uk/horrocks/Slides/index.html`, 2001.
16. DARPA Agent Markup Language Web Site. `www.daml.org`, 2001.
17. J. Smith. *UML Formalization and Transformation*. PhD thesis, Northeastern University, Boston, MA, December 1999.
18. 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.
19. J. Smith, M. Kokar, K. Baclawski, and S. DeLoach. UML formalization and transformation. In *ECOOP*, 2000.
20. J. Sowa, editor. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. PWS Publishing, 2000.
21. DAML+OIL Specification. `www.daml.org/2001/03/daml+oil-index.html`, 2001.