Softw Syst Model (2002) 1: 142–156 / Digital Object Identifier (DOI) 10.1007/s10270-002-0008-4

Extending the Unified Modeling Language for ontology development

Kenneth Baclawski[2], **Mieczyslaw K. Kokar**[2], **Paul A. Kogut**[1], **Lewis Hart**[5], **Jeffrey Smith**[3], **Jerzy Letkowski**[4], **Pat Emery**[5]

[1] Lockheed Martin Management and Data Systems, PO Box 8048, Philadelphia, PA 19101, USA
[2] Northeastern University, 360 Huntington Avenue, Boston, Massachusetts 02115, USA; E-mail: kenb@ccs.neu.edu
[3] Mercury Computer Systems, 199 Riverneck Road, Chelmsford, Massachusetts 01824-2820, USA
[4] Western New England College, 1215 Wilbraham Road, Springfield, Massachusetts 01119, USA
[5] AT&T Government Solutions Inc., 1900 Gallow Road, Vienna, Virginia 22182, USA

Initial sbmission: 16 February 2002/Revised submission: 15 October 2002
Published online: 2 December 2002 – © Springer-Verlag 2002

Abstract. There is rapidly growing momentum for web enabled agents that reason about and dynamically integrate the appropriate knowledge and services at runtime. The dynamic integration of knowledge and services depends on the existence of explicit declarative semantic models (ontologies). We have been building tools for ontology development based on the Unified Modeling Language (UML). 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. UML has many features, such as profiles, global modularity and extension mechanisms that are not generally available in most ontology languages. However, ontology languages have some features that UML does not support. Our paper identifies the similarities and differences (with examples) between UML and the ontology languages RDF and DAML+OIL. To reconcile these differences, we propose a modification to the UML metamodel to address some of the most problematic differences. One of these is the ontological concept variously called a property, relation or predicate. This notion corresponds to the UML concepts of association and attribute. In ontology languages properties are first-class modeling elements, but UML associations and attributes are not first-class. Our proposal is backward-compatible with existing UML models while enhancing its viability for ontology modeling. While we have focused on RDF and DAML+OIL 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.

Keywords: Ontology – Semantic web – Agents – OO modeling – UML – RDF – DAML

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 [30].

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 modeling 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) [13].
- 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 [30].

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) [21]. RDF is a recommendation within the XML suite of standards, developed under the auspices of the World Wide Web Consortium. RDF is developing quickly [10]. 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) [9, 17, 19, 20] is a language based on RDF and RDF Schema that will be able to express a much richer variety of constraints as well as support logical inference. For a good discussion of the design rationale of DAML+OIL see [18].

As in any data language, KR languages have the ability to express schemata 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* [12, 15, 16, 23]. An ontology is an explicit, formal semantic model. It is derived from the corresponding notion in Philosophy. See the classical work by Bunge [6, 7] as well as more recent work by Guarino, Uschold and their colleagues [14, 31]. Data conforming to an ontology is often referred to as an *annotation* or as *markup*, 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 [22]. These agents can reason about knowledge and can dynamically integrate services at run-time. Formal ontologies are the basis for such agents. DAML+OIL is designed to support agent communication and reasoning.

RDF and DAML+OIL, which currently do not have any standard graphical form, could leverage the UML graphical representation. In addition, RDF and DAML+OIL 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+OIL ontologies. Cranefield [8] has also been promoting ontology development using UML and has been translating UML to RDF. Although their purposes are different, UML and DAML+OIL have many characteristics in common. For example, both have a notion of a *class* which can have *instances*. Table 1 gives an attempt to capture the similarities between the two languages at a high level.

Our paper discusses the similarities and differences between UML and DAML+OIL and how the differences might be reconciled. One of the most problematic differences is the KR notion variously called a relation, predicate or property. This KR notion appears, at a first glance, to be the same as a UML association or attribute. This is misleading, since the corresponding KR concept is a first-class modeling element, while UML associations and attributes are not first-class. More precisely, in KR systems a relation can exist without specifying any classes that it might relate.

In fact, in some KR systems, it is the properties that are the primary modeling primitive, and classes are effectively relegated to a secondary status. See, for example, [25, 26, 32, 33]. In such a KR system, it is assumed that the universe consists of instances that have properties. A set of classes is then chosen so as to satisfy the criterion of "cognitive economy." This essentially means that the instances can be described economically from the point of view of memory usage. However, there is a trade-

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

DAML Concept	Similar UML Concepts
Ontology	Package
Class	Class
As Sets (union, intersection, etc.)	Not Supported
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

off between memory usage and time spent deriving consequences which might otherwise be represented explicitly.

In UML, on the other hand, an association is defined in terms of *association ends*, which must be related to classifiers, and an attribute is always within the context of a single class. This difference between UML and most other KR languages has also been noted by Cranefield [8]. We are proposing a modification to the UML metamodel to deal with this issue.

2 DAML background

The aim of the DAML program is to achieve "semantic interoperability between Web pages, databases, programs, and sensors." [9] 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." [9] DAML addresses the problem of building a set of ontologies based on commonly accepted 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. Toward this end, DAML+OIL 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+OIL enriches and formalizes metadata annotations (see Fig. 1).

DAML+OIL is only part of the *Semantic Web* vision [4,24] 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+OIL 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.

Both DAML+OIL and RDF express facts using *statements* (also called *triples*). An RDF statement consists of a *predicate*, *subject* and *object*. The predicate is required to be a property and the subject is required to be a resource (i.e., specified by a URI).

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 consistent [27–29]. In this section we discuss in general terms some of the issues that

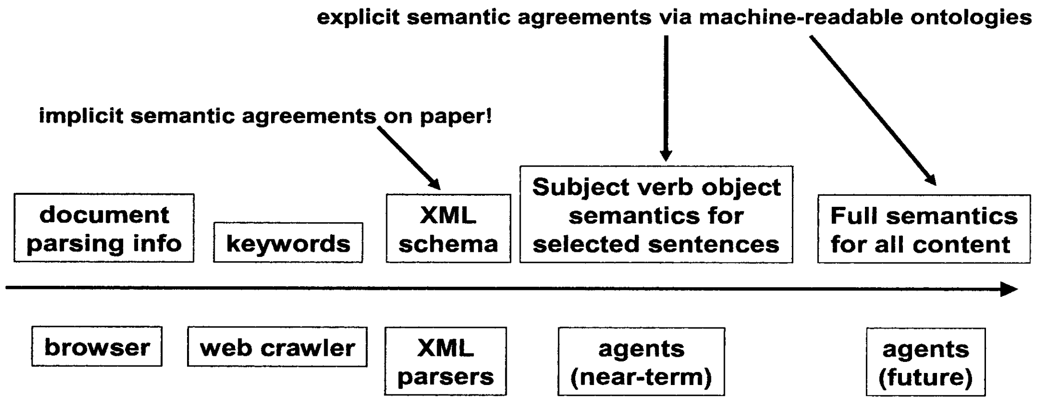

explicit semantic agreements via machine-readable ontologies
implicit semantic agreements on paper!
document parsing info
keywords
XML schema
Subject verb object semantics for selected sentences
Full semantics for all content
browser
web crawler
XML parsers
agents (near-term)
agents (future)

Fig. 1. The Evolution of Metadata

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 for the mapping to be consistent. The notion of mapping consistency will be discussed below.
– 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+OIL.

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 usually means 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+OIL 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 $f(M_2)$. The notion of a mapping that preserves semantic equivalence is substantially weaker than that of a mapping that preserves semantics (i.e., for every model M, the models M and $f(M)$ are semantically equivalent). The latter notion is what one usually means by a mapping being "correct." In the case of UML and DAML+OIL, the mapping is defined only on those UML models that are necessary for expressing DAML+OIL 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 semantic equivalence 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.

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 $\#f_1(M)$, $\#f_2(f_1(M))$, $\#f_1(f_2(f_1(M)))$, ... 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+OIL class and each association end to a DAML+OIL 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+OIL mapping

In order to discuss the similarities between UML and DAML+OIL an initial incomplete mapping between the languages has been created. Table 1 presents a high-level mapping of concepts from UML and DAML+OIL, and serves as an overview of the 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+OIL ontologies. Legacy class diagrams that were not originally intended for DAML+OIL applications would be usable for DAML+OIL purposes but would need modification in order to make full use of DAML+OIL capabilities. It is important to note that we make no claim that this mapping is "correct" or that it "preserves semantics." On the other hand, we have attempted to ensure that the mapping is consistent in the sense that it preserves semantic equivalence.

The mapping in Table 2 mentions a number of constructs that UML does not support. We discuss how UML may be extended to support them in Sect. 6 below. In principle, each of the "Not supported" entries in the table could be added to UML by introducing a suitable stereotype. This was the approach taken by DUET [1], and we make use of this for many of our examples. Such an approach is certainly sufficient for the goal of using UML as a user interface for DAML, but it does not define any semantics for the stereotypes. Without suitable constraints a mapping defined by using UML stereotypes need not be consistent. For example, the `sameClassAs` property of DAML+OIL should be constrained to link only classes. In Sect. 6 we address this problem by proposing a modification of UML that introduces appropriate constraints.

4.1 Representing subclass relationships

The distinction between the UML notion of generalization and the DAML+OIL concept of `subClassOf` is a good example of the difference between consistency and

Table 2. Mapping Between UML and DAML

UML	DAML
Class	Class
"classifier" role	type
type of ModelElement	type
attribute	ObjectProperty or DatatypeProperty
association end	ObjectProperty
specialization of classes	subClassOf
specialization of associations	subPropertyOf
note	comment
name	label
"seeAlso" tagged value on a class and association	seeAlso
"isDefinedBy" tagged value on a class and association	isDefinedBy
Not supported	"Restriction" class
Not supported	"complementOf" property
Not supported	"unionOf" property
Not supported	"intersectionOf" property
"onProperty" association	onProperty
"toClass" association	toClass
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
Not supported	equivalentTo
Not supported	sameClassAs
Not supported	samePropertyAs
Not supported	sameIndividualAs
Not supported	differentIndividualFrom
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
relationship between the association ends of an association	inverseOf
Not supported	TransitiveProperty

correctness. UML specialization/generalization, while not formally specified, is most commonly used to represent the subclass/superclass relationship in object-oriented programming languages. This relationship is fundamentally *behavioral*, while the DAML+OIL `subClassOf` property is formally defined as being *set-theoretic*. These two points of view are formally incompatible.

To see why these two are incompatible, consider the example of classes `Square` and `Rectangle` defined as follows:

```
class Square {
  double length;
  Square(double l) { length = l; }
  double area() { return length * length; }
  void magnify(double scale) { length =
                                scale * length; }
}
class Rectangle {
  double length, width;
  Rectangle(double l, double w) { length = l;
                                     width = w; }
  double area() { return length * width; }
  void magnify(double scale) { length *=
                          scale; width *= scale; }
  void stretchLength(double s) { length *= s; }
  void stretchWidth(double s) { width *= s; }
}
```

From a set-theoretic perspective, a square can only be a subclass of rectangle because the set of squares is a subset of the set of rectangles. However, this point of view ignores the behavioral aspects of squares and rectangles. Indeed, in most object-oriented programming languages one could not easily specify the `Square` class above as a subclass of `Rectangle`. Any such attempt would conflict with the principle of substitutability: one cannot regard a square object as a rectangle because a square cannot be stretched, whereas a rectangle with length equal to the width can be stretched. On the other hand, from a behavioral perspective, a rectangle can be a subclass of a square. Indeed, one could define it as follows:

```
class Rectangle extends Square {
  double width;
  Rectangle(double l, double w) { super(l);
                                   width = w; }
  double area() { return length * width; }
  void magnify(double scale) { length *= scale;
                               width *= scale; }
  void stretchLength(double s) { length *= s; }
  void stretchWidth(double s) { width *= s; }
}
```

Note the use of overriding for both the `area` and `magnify` methods. Using specialization/generalization as in this example (i.e., adding additional attributes and methods, and overriding methods) is far more common than the set-theoretic point of view (i.e., a subclass that restricts the superclass without adding any attributes or methods).

This example has generated considerable discussion, including many who dismiss the possibility that `Rectangle` might be a subclass of `Square` as being *unnatural* or even *bizarre*. See [3] for a refutation of such arguments.

From this example, it is clear that UML generalization is semantically quite different from the DAML+OIL `subClassOf` property. Yet, using the former to specify the latter preserves semantic equivalence, because any two classes such that one is a specialization of the other will be mapped to two DAML+OIL classes that are related by the `subClassOf` relation. Accordingly, mapping UML generalization to DAML+OIL `subClassOf` is consistent.

4.2 Representing DAML+OIL 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+OIL the property "mother" exists independently of the two classes, and any object could be a mother unless suitable restrictions are imposed. For example, in Fig. 2, this is represented as a restriction for class Person, on property "mother", to class Woman. Other restrictions could also be imposed to represent other notions of motherhood (animals, companies and so on).

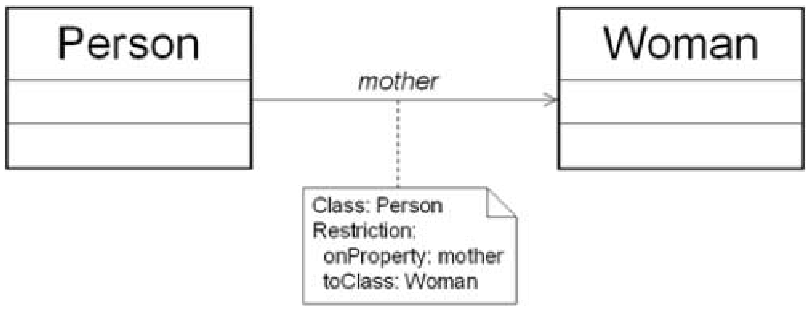


Fig. 2. DAML+OIL Property Restriction

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

Fig. 3. DAML+OIL Translation of Fig. 2

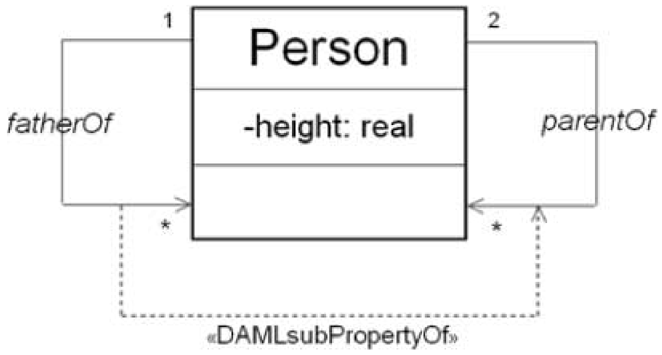


Fig. 4. Example of a SubProperty

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

Another concept of the mapping can be seen in Fig. 4, which shows one of the UML representations of a DAML+OIL `subProperty`. 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.

4.3 Representing DAML+OIL instances

Figure 5 illustrates the concept of an instantiated class in UML. In a similar fashion, this would be described in DAML+OIL as an element identified as "Tommy", with type identified as Person and the value "53" assigned to the property "height".

4.4 Representing facets of properties

To demonstrate the mapping between UML multiplicity and DAML+OIL cardinality, Fig. 6 depicts the correspondance between the multiplicity of an association end and the corresponding cardinality in DAML+OIL. 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.

Figures 7 and 8 depict special cases of DAML+OIL properties with predefined cardinality restrictions. The first of these is called an Unambiguous Property and is depicted in Fig. 7. An Unambiguous Property is defined in DAML+OIL 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+OIL 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

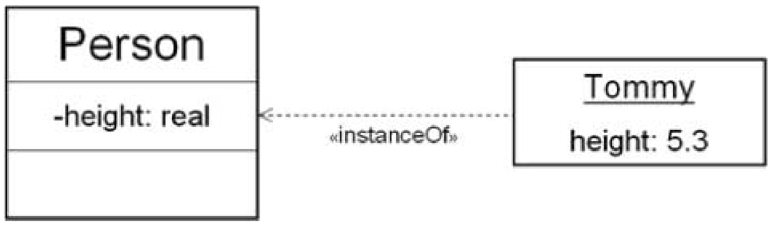


Fig. 5. DAML+OIL Type Property

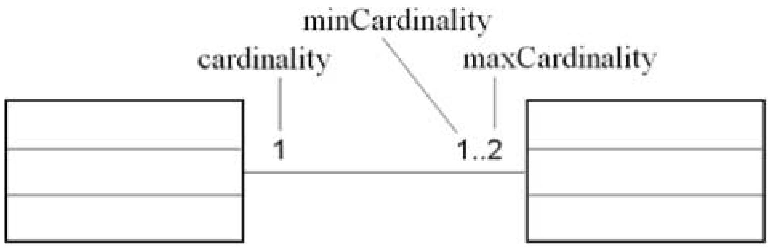


Fig. 6. DAML+OIL Cardinality

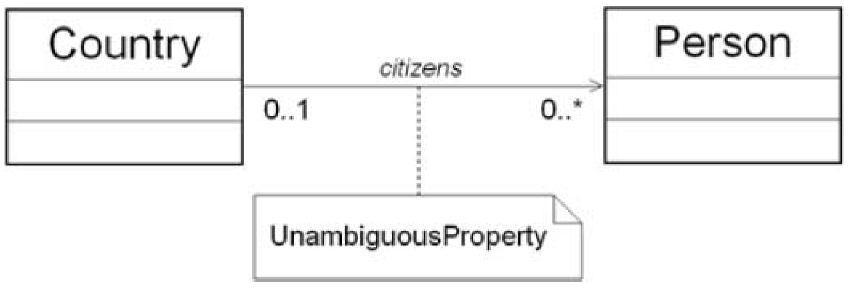


Fig. 7. Example of an Unambiguous Property

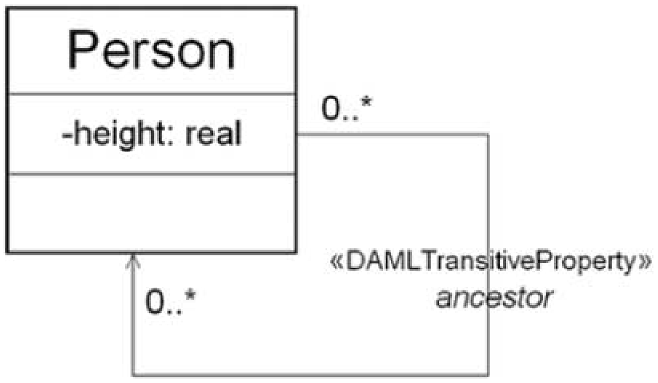


Fig. 8. Example of a Transitive Property

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 KR languages

While there are many similarities between UML and KR languages such as RDF and DAML+OIL, there are also many differences. Reconciling these differences has been one of the major problems of our project. We now discuss some of the major incompatibilities.

5.1 Open and closed worlds

An important distinction between KR approaches and object-oriented approaches is the notion of *monotonicity.* A logical system is monotonic if adding new facts can never cause previous facts to be falsified. Of course, one must be careful to define which facts are being considered in this process so that it makes sense. Both RDF and DAML+OIL are monotonic: asserting a new fact can never cause a previously known fact to become false.

By contrast, UML and other OO systems are typically not monotonic. There are many forms of nonmonotonic logic, but the one that is closest to UML and OO systems is a logic that assumes a *closed world.* A simple example can illustrate how monotonicity affects inference. Suppose that one specifies that every person must have a name. Consider what would happen if a particular person object does not have a name. In UML this situation would be considered to be a violation of the requirement that every person must have a name, and a suitable error message would be generated. In a monotonic logic, on the other hand, one cannot make any such conclusion. The person who appears not to have a name really does have one, it just isn't known yet.

As another example, 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 multiplicity 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 and that would be the end of it. In a monotonic logic, on the other hand, one cannot make such a conclusion, at least not directly. It is possible that `Computer Science` and `Chemistry` are the same department, one just hasn't yet seen the statement that they are the same. In DAML+OIL one could assert that two resources (in this case `Computer Science` and `Chemistry`) are the same by using the `equivalentTo` property. It is unlikely that the annotator intended this, but it is consistent nonetheless.

Virtually all of the consistency constraints that one is accustomed to impose with UML (including domain constraints, range constraints and multiplicity constraints) have very different consequences in monotonic and nonmonotonic logics. This distinction between UML and KR representations would therefore seem to be insurmountable. Indeed, it does make it effectively impossible to define a mapping between these languages that preserves semantics. However, it does not prevent one from defining a mapping that preserves semantic equivalence, and that is all that we are attempting to achieve.

5.2 Metalevels

Unlike most data modeling languages, KR languages do not have a rigid separation between meta-levels. Indeed, they often have none at all. 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 in UML goes between two metalevels. In RDF, as in many other KR languages, instances of a class may also be classes, and a chain of "instanceOf" links may be of any length. The RDF `Class` entity, in particular, is an instance of itself. While DAML+OIL inherits some aspects of RDF, the model-theoretic semantics for DAML+OIL differs from RDF in choosing to make a clear distinction between metalevels.

5.3 Modularity

RDF and DAML+OIL as well as most other KR languages do not have profiles, packages or any of the other modularity mechanisms supported by UML and UML-based CASE tools. RDF and DAML+OIL does make use of XML namespaces, but only for disambiguating names, not as a package mechanism. DAML+OIL introduced a notion of an `Ontology` and an `imports` property. However, very little is mentioned about what these mean, and what is mentioned is informal. For example, there is no requirement that one can only import an ontology, and there is no semantic distinction between a statement or resource being in one ontology rather than being in another.

5.4 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+OIL 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. However, lists and ordered associations are semantically different. UML has a package notion which can be used as a container for model elements. However, UML packages are not at the same metalevel as RDF containers and DAML+OIL lists, and the presentation features of packages also make them unsuitable for representing RDF containers.

5.5 Property

As we have noted in Sect. 1 above, the DAML+OIL notion of property is a first-class modeling element, while the UML notion of an association is not first-class. Furthermore, a UML binary association always has just one domain class and one range class. DAML+OIL properties are not limited in this manner, and one can specify complex domain and range constraints involving many classes.

Another significant difference between UML and DAML+OIL is that the relation between UML classes and associations is not exactly the same as between DAML+OIL 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+OIL, 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+OIL is that descriptions of both classes and properties in DAML+OIL can be distributed over various Web sites. 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. On the other hand, the idea of a property being associated with multiple classes is more flexible and might foster reuse. 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+OIL, they would all be the same property.

For instance, the `location` property could be used to 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 must be modeled as a second association. These two associations are necessarily different even if they use the same name because the associations are in different namespaces.

In RDF and DAML+OIL (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.

To deal with the problem of first-class properties, we propose that a new type of model element be added to UML for representing properties. Since RDF properties are unidirectional, it would be inconsistent to view a property as a grouping of associations. To be consistent one must interpret a property as an aggregation of association *ends* from different associations. This is discussed in more detail in Sect. 6.

5.6 Class constructors

One feature that DAML+OIL introduced is the ability to construct classes using boolean operations (union, intersection and complement) and quantifiers. The boolean operations were introduced using list-valued properties called `unionOf` and `intersectionOf`, along with a class-valued property named `complementOf`. Quantification was introduced by using the notion of a *restriction*.

The only difficulty with introducing the same capability in UML is that there is no suitable notion of a container or list in UML as noted in Sect. 5.4 above. Introducing a notion of a list just for this purpose seems unnecessary inasmuch as the DAML+OIL use of lists in this case does not use the ordering information.

The feature of lists that DAML+OIL does use is the fact that lists are *bounded* containers. A container is bounded if one cannot add, remove or change elements in the container by simply asserting new facts. A list is bounded because any alteration in the list requires that some fact about the list be unasserted. This feature of lists is important only for a monotonic logic discussed in Sect. 5.1 above. If one assumes a closed world, as one typically does in UML models, then boundedness is not useful. Furthermore, there are other mechanisms in UML for specifying that a feature of a model is not changeable. Accordingly, our proposal for introducing class constructors into UML does not make use of lists.

5.7 Cardinality constraints

UML multiplicity constraints can be consistently mapped to DAML+OIL cardinality constraints as in Fig. 6. The only incompatibilities are the result of properties being first-class model elements that 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+OIL 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+OIL, one must introduce an inverse property in order to specify a cardinality constraint on the domain of a property.

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 *GeneralizableElement*, 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+OIL 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+OIL are linked together either through the use of URIs or by using the hierarchical containment relationship of XML. DAML+OIL 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 model elements 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+OIL 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.

6 Recommendations

We now give a specific recommendation for a modification to the UML metamodel that would enable one to model using first-class properties as well as to construct classifiers using boolean operations and quantification. The classifier constructors enable one to specify complex crosscutting constraints using UML. The metamodel for our proposal is shown in Figures 9 and 10. All of the metaclasses in these figures are already part of UML except for `Property`, `Restriction`, `Union`, `Intersection` and `Complement`. We first give the rationale and semantics of the `Property` metaclass and then consider the classifier constructors.

We make no claim that this is a complete specification. Our intention was to raise issues and to outline a solution. A full OMG proposal is beyond the scope of this paper. Such a proposal should consider other crosscutting aggregations of model elements. For example, one could

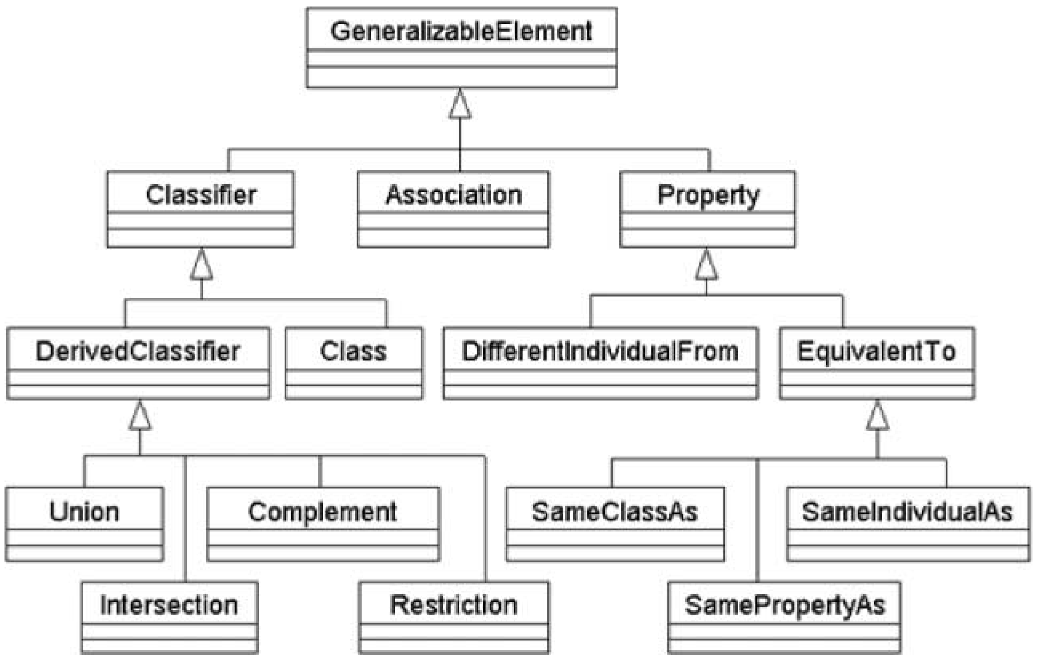


Fig. 9. The Specification of the Proposed UML Modification: Metaclass Hierarchy

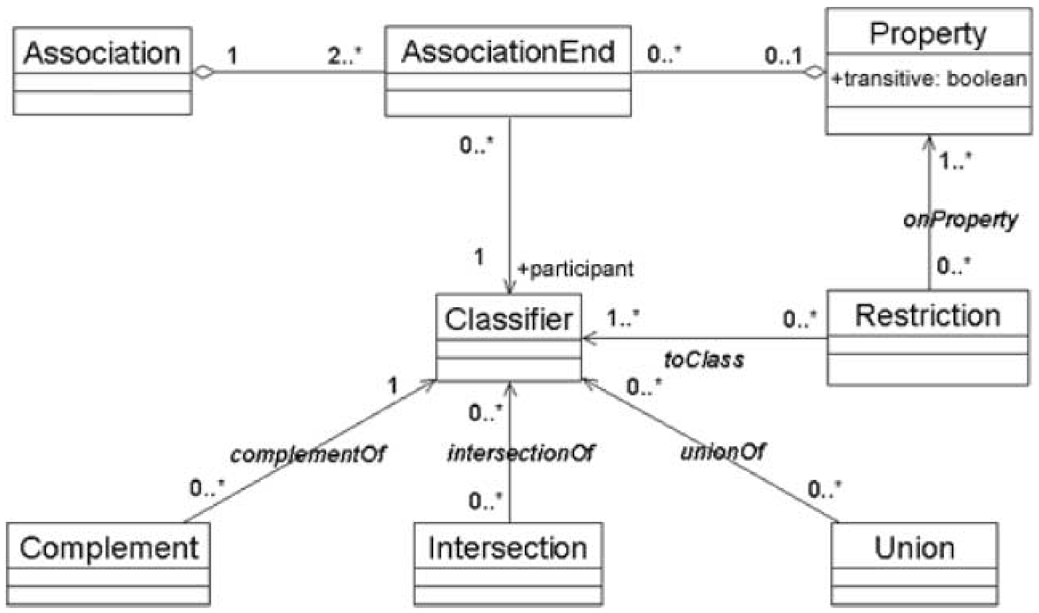


Fig. 10. The Specification of the Proposed UML Modification: Meta-Associations

aggregate attributes and methods in much the same way that association ends are aggregated by using properties.

6.1 Property recommendation

DAML+OIL 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, [2]). In DAML+OIL a predicate is represented by a property. However, the DAML+OIL 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 if it does not break any existing models. In addition, the programming language community has introduced a notion of first-class properties called *(crosscutting) aspects* [11]. Support for modeling aspects would also be desireable. We argue here that by adding a few additional model elements to the UML metamodel one can make UML compatible with knowledge representation languages as well as help support aspect-oriented programming.

To close the gap in the expressibility of UML, we propose to modify UML by adding a metaclass called *Property* as in Figures 9 and 10. 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. The fact that `Property` is a first-class concept is shown by the fact that `Property` can exist without being associated with any classes. A property can be constrained by using restrictions. The notion of a `Restriction` is discussed in more detail in the next section.

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. 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+OIL and UML, then the resulting mapping is unbounded, as has been discussed in Sect. 5.5.

The semantics for metamodel elements is specified in UML by using OCL. We now give an example of such a specification, including both the rationale for it and the formal expression. A property is a grouping of association ends. Properties "crosscut" 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 generalizable elements 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 a property can only be specializations or generalizations of other properties.

6.2 Class constructor recommendation

We propose a mechanism in UML whereby one can construct classifiers from other classifiers using the same boolean operations and quantifiers as in DAML+OIL. The proposed mechanism differs somewhat from DAML+OIL because UML does not have a suitable notion of a container as noted in Sect. 5.4 above. The metamodel for our proposal is shown in Figures 9 and 10. The metaclasses `Union`, `Intersection` and `Complement` play the roles of the DAML+OIL properties `unionOf`, `intersectionOf` and `complementOf`, respectively. Although the first two properties are list-valued, DAML+OIL does not make any use of the list ordering. The `Restriction` metaclass is almost the same as the DAML+OIL notion of a metaclass. We only show the case of a `toClass` restriction in Fig. 10. A complete proposal would also include the other forms of restriction specified in DAML+OIL: `hasValue`, `hasClass`, `minCardinality`, etc.

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 (via `onProperty`), 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 properties, the Restriction classifiers can only be generalizations and specializations of other Restriction classifiers.

6.3 Examples of the proposed constructs

To illustrate how one would use the proposed modeling constructs, we present an example in Fig. 11. In this example, there is an association between `Faculty` and `Organization`, one of whose roles (association ends) is named `affiliation`. We specify that this association end belongs to the `location` property by appending the property name to the association end name, separated by a colon. The property itself is defined elsewhere. In this example, it is shown using the graphical symbol for a class, modified by using the `Property` stereotype.

The classifier constructors are also shown using the same class graphical symbol, modified with the appropriate stereotype. The `Union` classifier specifies the union of all schools, universities and institutes that are not companies. The `Restriction` specifies a classifier whose objects are the ones whose `location` is a school, university or non-company institute. Because the `Faculty` class is a specialization of the `Restriction`, the affiliation(s) of a faculty member must be organizations that are schools, universities or non-company institutes.

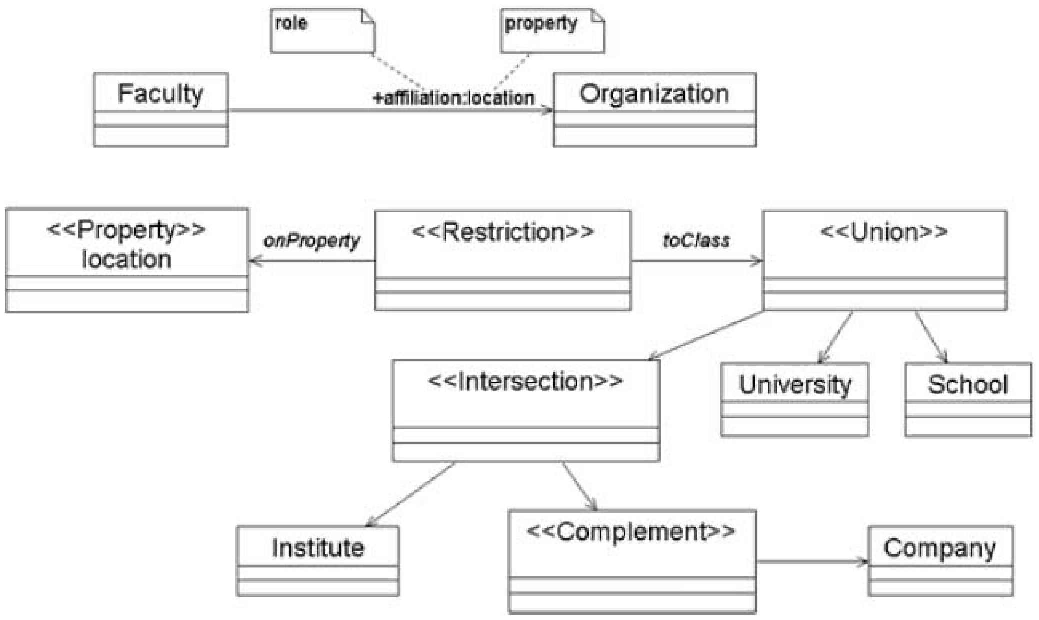


Fig. 11. Example Model using the Proposed UML Extension

7 Conclusion

In this paper we have reported on our work in progress on using UML as an ontology development environment. We have identified similarities and differences between UML and KR languages, and we have discussed how they can be mapped to each other in the case of DAML+OIL. In the "similarities" discussion we showed how UML concepts can be mapped to DAML+OIL. 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+OIL 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+OIL `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 modification to the UML metamodel. We also introduced the ability to construct classifiers using boolean operations and quantification. These are useful for specifying crosscutting constraints that can be imposed on many associations all at once. The motivation for these modifications to UML was to make it more compatible with KR languages and aspect-oriented programming. This might lead to the acceptance of UML by the knowledge representation community as the preferred graphical notation for KR languages, such as DAML+OIL, 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

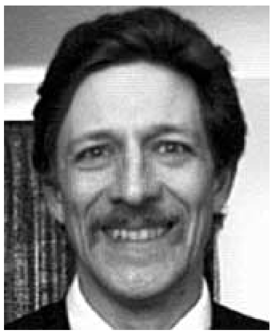
1. DAML UML enhanced tool (DUET), 2002. `http://grcinet.grci.com/maria/www/CodipSite/Tools/Tools.html`
2. Abbott, R.: Program design by informal English descriptions. Comm. ACM, 26(11): 882–894, 1983
3. Baclawski, K., Indurkhya, B.: The notion of inheritance in object-oriented programming. Comm. ACM, 37(9): 118–119, September 1994
4. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American, May 2001
5. Booch, G., Jacobson, I., Rumbaugh, J.: OMG Unified Modeling Language Specification, March 2000 Available at: `www.omg.org/technology/documents/formal/unified_modeling_language.htm`.
6. Bunge, M.: Treatise on basic philosophy. III: Ontology: The furniture of the world. Reidel, Dordrecht, 1977
7. Bunge, M.: Treatise on basic philosophy. IV: Ontology: A world of systems. Reidel, Dordrecht, 1979
8. Cranefield, S.: Networked knowledge representation and exchange using UML and RDF. J. of Digital information, 1(8), February 2001
9. DAML. DARPA Agent Markup Language Web Site, 2001, `www.daml.org`
10. Decker, S., Brickley, D., Saarela, J., Angele, J.: A query and inference service for RDF. In: QL'98 – The Query Language Workshop. W3C, 1998
11. Elrad, T., Filman, R., Bader, A.: Aspect-oriented programming: Introduction. Comm. ACM, 44(10): 29–32, October 2001
12. Falkenberg, E.D., Lyytinen, K., and Verrijn-Stuart, A.A. (eds.): Ontological Evaluation of the OML Metamodel, IFIP Conference Proceedings, vol. 164. Kluwer, 2000
13. Genesereth, M.: Knowledge Interchange Format draft proposed American National Standard (dpANS) NCITS.T2/98-004, 1998. Available at: `logic.stanford.edu/kif/dpans.html`
14. Guarino, N., Giaretta, P.: Ontologies and knowledge bases: Towards a terminological clarification. In: Mars, N. (ed.): Towards Very Large Knowledge Bases. IOS Press, 1995
15. Heflin, J., Hendler, J., Luke, S.: Coping with changing ontologies in a distributed environment. In: AAAI-99 Workshop on Ontology Management. MIT Press, 1999

16. Heflin, J., Hendler, J., Luke, S.: 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
17. Hendler, J., McGuinness, D.: The DARPA Agent Markup Language. IEEE Intelligent Systems, 15, No. 6: 67–73, 2000
18. Horrocks, I.: DAML+OIL Design Rationale, 2001. www.cs.man.ac.uk/horrocks/Slides/index.html
19. Horrocks, I., Patel-Schneider, P., vanHarmelen, F.: Reviewing the design of DAML+OIL: An ontology language for the semantic web. In: AAAI/IAAI 2002, 2002, pp. 792–797
20. McGuinness D. L., Fikes, R., Hendler, J., Stein, L.A.: DAML+ OIL: An ontology language for the semantic web. IEEE Intelligent Systems, September/October: 72–80, 2002
21. Lassila, O., Swick, R.: Resource description framework (RDF) model and syntax specification, Feburary 1999. www.w3.org/TR/REC-rdf-syntax
22. McGuinness, D.: Ontologies and online commerce. IEEE Intelligent Systems, 16(1): 8–14, 2001
23. McGuinness, D.L., Fikes, R., Rice, J., Wilde, S.: An environment for merging and testing large ontologies. In: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000), 2000
24. Miller, E., Swick, R., Brickley, D., McBride, B.: Semantic web activity page, April 2001. Available at: www.w3.org/2001/sw/
25. Parsons, J., Wand, Y.: Choosing classes in conceptual modeling. Comm. Assoc. Computing Machinery, 40(6): 63–69, 1997
26. Parsons, J., Wand, Y.: Using objects for systems analysis. Comm. Assoc. Computing Machinery, 40(12): 104–110, 1997
27. Smith, J.: UML Formalization and Transformation. PhD thesis, Northeastern University, Boston, MA, December 1999
28. Smith, J., Kokar, M., Baclawski, K.: Formal verification of UML diagrams: A first step towards code generation. In: Eighth OOPSLA Workshop on Behavioral Semantics, November 1999, pp. 206–220
29. Smith, J., Kokar, M., Baclawski, K., DeLoach, S.: Category theoretic approaches of representing precise UML semantics. In: Proceedings of the Precise UML Workshop at ECOOP 2000, Sophia Antipolis, France, 2000
30. Sowa, J. (ed.): Knowledge Representation: Logical, Philosophical, and Computational Foundations. PWS Publishing, 2000
31. Uschold, M., Gruninger, M.: Ontologies: Principles, methods and applications. Knowledge Engineering Review, 11(2): 93–155, June 1996
32. Wand, Y., Storey, V., Weber, R.: An ontological analysis of the relationship construct in conceptual modeling. Trans. Database Systems, 24(4): 494–528, 1999
33. Wand, Y., Weber, R.: An ontological model of an information system. Trans. Software Engineering, 16(11): 1282–1292, 1990

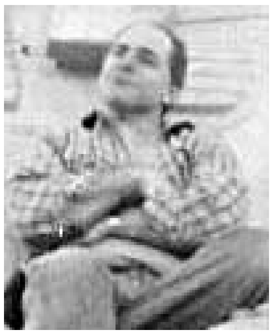
Kenneth Baclawski is an Associate Professor of Computer Science at Northeastern University. He has a B.S. from the University of Wisconsin and a Ph.D. from Harvard University. Prof. Baclawski's research interests include Formal Methods in Software Engineering, High Performance Knowledge Management, and Database Management. He teaches a variety of courses in software engineering, object-oriented systems, databases, data modeling and operating systems. He has participated in and directed many large research projects funded by government agencies including the NSF, DARPA and the NIH. He holds five U.S. patents. Prof. Baclawski is a member of the ACM and the IEEE. He has held many positions in both organizations, including President of the Boston Chapter of the ACM.

Mieczyslaw M. Kokar is with the Department of Electrical and Computer Engineering at Northeastern University in Boston. His technical research interests include formal methods in software engineering, self-controlling software, and information fusion. Dr. Kokar teaches graduate courses in software engineering, formal methods, artificial intelligence, and software engineering project. Dr. Kokar's research has been supported by DARPA, NSF, AFOSR and other agencies. He has an M.S. and a Ph.D. in computer systems engineering from Technical University of Wroclaw, Poland. He is a member of the IEEE and of the ACM.

Paul Kogut is the principal investigator for the Lockheed Martin team in the DARPA DAML program. He has participated in a variety of research projects related to agents, natural language processing and knowledge representation. He was a resident affiliate at the Carnegie Mellon University Software Engineering Institute in 1994. As an employee of the U.S. Army CECOM in the 1980s, he provided software engineering support for Army, Air Force and Marine C2 systems. Dr. Kogut is an adjunct professor at Penn State Great Valley where he teaches an AI course that emphasizes multi-agent systems. He has a Ph.D. in Computer Science from Lehigh University.



Mr. Hart is the Chief Scientist for Intelligent Agent Systems at AT&T Government Solutions, Inc, in Vienna, Virginia. He is currently the principal investigator on the Components for Ontology Driven Information Push (CODIP) project, a part of the DARAP Agent Markup Language (DAML) program. He is also co-inventor of MARIA, a distributed agent architecture. Mr. Hart's research interests involve the application of software agent, artificial intelligence and knowledge-based technologies to create intelligent software systems. He received an MS in Industrial and Systems Engineering and BS in Physics both from the University of Florida at Gainesville.



Dr. Jeff Smith has a Ph.D. from Northeastern University in Computer Systems Engineering, an M.S. in Engineering Management from Southern Methodist University, an M.S. in Computer Science from East Texas State University and a B.S. in Computer Systems Engineering from the University of Massachusetts. Dr. Smith joined Mercury Computers in April of 2000 as a Consulting Scientist. He has contributed to modeling and programming Mercurys high performance, streaming, data flow machines. Dr. Smith has been leading a team to develop programs to establish Software Radio as a key Mercury business area. He is also one of 3 co-chairs of the OMG Software Radio DSIG. Prior to joining Mercury, Dr. Smith had a 23 year engineering track record in the acquisition, management, research and development of Advanced Development/Technology Programs.

Jerzy Letkowski received his Ph.D. in Management and Information Sciences from the Wroclaw University of Technology (WUT), Poland, in 1977. He has been engaged in academic work at WUT between 1994 and 1984 and at Western New England College (WNEC), Springfield, Massachusetts, since 1985. His research priority is to engage in scholarship that enriches teaching computer programming and quantitative analysis as well as helps develop better WWW tools and applications. His broad interdisciplinary interests are reflected in publications and conference presentations. More information is available at `http://wnec.edu/∼jletkows`.

Patrick M. Emery As a Principal Software Systems Engineer with AT&T for over eleven years, Patrick M. Emery specializes in research designing and developing intelligent agents using Internet standards and object oriented techniques. Mr. Emery has applied his considerable experience in software engineering, system administration, database design, artificial intelligence, and information security to making systems work.