

OWL Full and UML 2.0 Compared

This document is intended to establish the relationship between the relevant features of UML 2.0 and OWL as part of the development of the Ontology Definition Metamodel (ODM). The contents of the final version of this document will be incorporated in the ODM specification to provide guidelines for the translation of UML models to the ODM.

Version 2.4
12 March, 2004

Co-submitters:

<i>AT&T/Gentleware</i>	Lewis Hart, Patrick Emery
<i>DSTC</i>	Bob Colomb, Kerry Raymond, Sarah Taraporewalla
<i>IBM</i>	Dan Chang, Yiming Ye
<i>Sandpiper Software</i>	Elisa Kendall, Mark Dutra

CONTENTS

1. INTRODUCTION	3
2. FEATURES IN COMMON (MORE OR LESS).....	4
2.1 UML KERNEL	4
2.2 CLASS AND PROPERTY - BASICS	6
2.3 MORE ADVANCED CONCEPTS.....	10
2.4 SUMMARY OF MORE OR LESS COMMON FEATURES	15
3. OWL BUT NOT UML.....	16
3.1 PREDICATE DEFINITION LANGUAGE	16
3.2 NAMES	17
3.3 OTHER OWL DEVELOPMENTS	17
4. IN UML BUT NOT OWL	18
4.1 BEHAVIORAL FEATURES.....	18
4.2 COMPLEX OBJECTS.....	18
4.3 ACCESS CONTROL	19
4.4 KEYWORDS	19
5. REFERENCES.....	19

1. Introduction

This note compares the features of OWL Full (as summarized in [1]) with the features of UML 2.0 [2] as a preliminary analysis supporting the design of an Ontology Development Metamodel. It first looks at the features the two have in common, although sometimes represented differently, then the features in one but not the other. Little attempt is made to distinguish the features of OWL Lite or OWL DL from those of OWL Full. This note ignores secondary features such as headers, comments and version control. In the features in common, a sketch is given of the translation from a model expressed in UML to an OWL expression. In several cases, there are alternative ways to translate UML constructs to OWL constructs. This document selects a particular way in each case, but the translation is not intended to be normative. In particular applications other choices may be preferable.

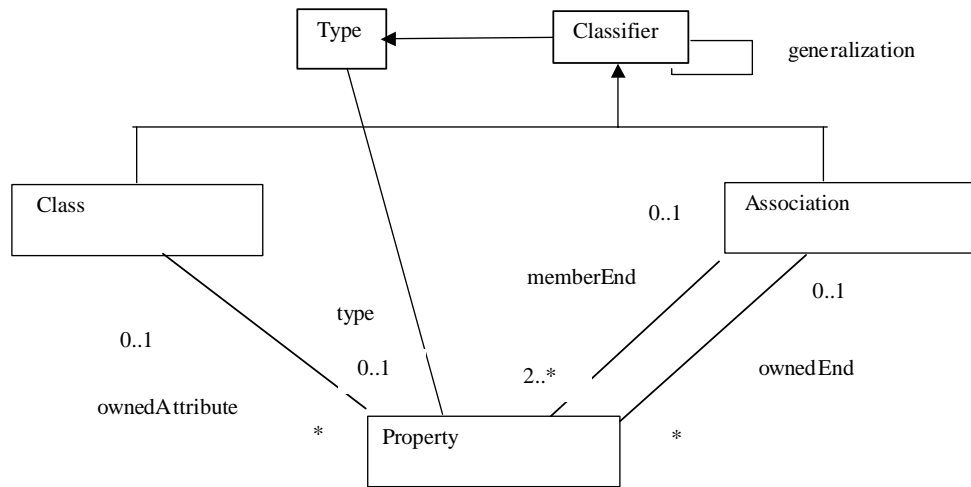
The possible translation of OWL to UML is out of scope of this document.

UML models are organized in a series of metalevels : M3, M2, M1 and M0, as follows:

- M3 is the MOF, the universal modeling language in which modeling systems are specified.
- M2 is the model of a particular modeling system. The UML metamodel is an M2 construct, as it is specified in the M3 MOF.
- M1 is the model of a particular application represented in a particular modeling system. The UML Class diagram model of an order entry system is an M1 construct expressed in the M2 metamodel for the UML Class diagram.
- M0 is the population of a particular application. The population of a particular order entry system at a particular time is an M0 construct.

2. Features in common (more or less)

2.1 UML Kernel



Abstracted from UML Superstructure [2] Figure 30, Section 7.11 page 80

Figure 1. Key aspects of UML Class Diagram

The structure of UML is formally quite different from OWL. What we are trying to do is to understand the relationship between an M1/M0 model in UML and the equivalent model in OWL, so we need to understand how the M1 model is represented in the M2 structure shown. First, a few observations from Figure 1.

- Most of the content of a UML model is in the M1 specification. The M0 model can be anything that meets the specification of the M1 model.
- There is no direct linkage between Association and Class. The linkage is mediated by Property.
- A Property is a structural feature (not shown), which is typed. The M1 model is built from structural features.
- Both Class and Association are types.
- A class always has a property which is the structural feature that implements it.
- A property may or may not be owned by one or more classes. A property owned by at least one class is called *navigable*¹. A property owned by no class is called *not navigable*². Only binary associations can have navigable ends.

It will help if we represent a simple M1 model in this structure (Figure 2).

¹ Called a member end in the Classes diagram of the UML superstructure

² Called an owned end in the Classes diagram of the UML superstructure

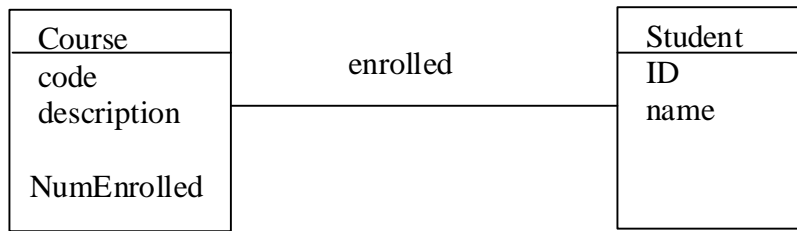


Figure 2. Simple M1 Model

The properties with their types are

Table 1

Property	Type
code	CourseIdentifier
description	string
NumEnrolled	integer
ID	StudentIdentifier
name	string

The classes are: Course, Student

Classes are represented by sets of *ownedAttribute* properties:

Table 2

Class	Owned Properties
Course	code, description, NumEnrolled
Student	ID, name

Associations are: enrolled

The association can be modeled in a number of different ways, depending on how classes are represented. If classes are represented as in table 2, one way is as the disjoint union of the owned attributes of the two classes.

Table 3

Association	Representation
enrolled	code, description, NumEnrolled, ID, name

But there are other ways to represent a class. If it is known that the property *code* identifies instances of *Course* and that the property *ID* identifies instances of *Student*, then an alternative representation of *enrolled* is

Table 4

Association	Representation
enrolled	code, ID

In this case, the properties *code* and *ID* would be of type *Course* and *Student* respectively.

2.2 Class and property - basics

Both OWL and UML are based on classes. A **class** is a set of **instances**. The set of instances associated at a particular time with a class is called the class' **extent**. But there are subtle differences.

In UML the extent of a class is an M0 object consisting of instances. An instance consists of a set of slots each of which contains a value drawn from the type of the property of the slot. The instance is associated with one or more classifiers. An instance of the class *Course* might be

Table 5

Classifier	code	title	NumEnrolled
Course	INFS3101	Ontology and the Semantic Web	0

But the M0 implementation of a class is not fully constrained. An equally valid instance of *Course* would be the name *INFS3101*, if it were decided that that name would identify an instance of the class. The remainder of the slots could be filled dynamically from other properties of the class.

In OWL, the extent of a class is a set of individuals, which are represented by names. Individual is defined independently of classes. There is a universal class *Thing* whose extent is all individuals in a given OWL model, and all classes are subclasses of *Thing*. The main difference between UML and OWL in respect of instances is that in OWL an individual may be an instance of *Thing* and not necessarily any other class, so could be outside the system in a UML model.

An OWL class is declared by assigning a name to the relevant type. For example

```
<owl:Class rdf:ID="Course"/>
```

An individual is at bottom an RDFS resource, which is essentially a name, so the individual INFS3101 will be declared with something like

```
<owl:Thing rdf:ID = "INFS3101"/>
```

Relationships among classes in OWL are called **properties**. That the class *course* has the relationship with the class *student* called *enrolled*, which was represented in the UML model as the association *enrolled*, is represented in OWL as a property

```
<owl:ObjectProperty rdf:ID = "enrolled"/>
```

Properties are not necessarily tied to classes. By default, a property is a binary relation between *Thing* and *Thing*.

So, in order to translate the M1 model of Figure 2 to OWL, UML Class goes to owl:Class.

Table 6

Class	Owned properties	OWL equivalent
Course	code, description, NumEnrolled	<owl:Class rdf:ID="Course"/>
Student	ID, name	<owl:Class rdf:ID="Student"/>

The relationships among classes represented in OWL by owl:ObjectProperty and owl:DatatypeProperty come from two different sources in the UML model. One source is the M2 association *ownedAttribute* between Class and Property, which generates the representation of a class as a bundle of owned properties as in Table 2. A M1 instance of *Class ownedAttribute Property* would translate as properties whose domain is *Class* and whose range is the type of *Property*. The UML *ownedAttribute* instance would translate to owl:ObjectProperty if the type of *Property* were a UML Class, and owl:DatatypeProperty otherwise. The translation of Table 2 is shown in Table 7. Note that UML *ownedAttribute* M2 associations are distinct, even if *ownedAttributes* have the same name associated with different classes. The owl property names must therefore be unique. One way to do this is to use a combination of the class name and the owned property name. Note also that since instances of *ownedAttribute* are always relationships among types, the equivalent OWL properties all have domain and range specified.

An alternative way to give domain and range to OWL properties is to use restriction to allValuesFrom the range class when the property is applied to the domain class. This is probably a more natural OWL specification. However, since all OWL properties arising from a UML model are distinct, the method employed in this document is adequate. Should a translation of a UML model be intended as a base for further development in OWL, an appropriate translation can be employed.

Table 7

Class	Owned property	Type of owned property	OWL equivalent
Course	code	CourseID	<owl:ObjectProperty rdf:ID="CourseCode"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="CourseID"/> </owl:ObjectProperty>
	description	string	<owl:DatatypeProperty rdf:ID="CourseDescription"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="http://www.w3.org/2001/ XMLSchema#string"/> </owl:DatatypeProperty>
	NumEnrolled	integer	<owl:DatatypeProperty rdf:ID="CourseEnrolled"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="http://www.w3.org/2001/ XMLSchema#integer"/> </owl:DatatypeProperty>
Student	ID	StudentIdent	<owl:ObjectProperty rdf:ID="StudentID"> <rdfs:domain rdf:resource="Student"/> <rdfs:range rdf:resource="StudentIdent"/> </owl:ObjectProperty>
	name	string	<owl:DatatypeProperty rdf:ID="StudentName"> <rdfs:domain rdf:resource="Student"/> <rdfs:range rdf:resource="http://www.w3.org/2001/ XMLSchema#string"/> </owl:DatatypeProperty>

Note that the translation in Table 7 assumes that a single name is an identifier for instances of the corresponding class. This is not always true. That is there are cases in which a relational database implementation would use a compound key to identify an instance of a class. Since OWL individuals are always unitary names, the translation of the UML class would construct a unitary name from the instances of the individual properties. For example, if the association *enrolled* were treated as a class (UML association class), its representing property might be a concatenation of Course.code and Student.id, so that student 1234 enrolled in course INFS3101 might be translated to an OWL individual with name 1234.INFS3101.

The second source of owl properties in a UML M1 model is the M1 population of the M2 class *association*. A binary UML association translates directly to an owl:ObjectProperty. The translation of Table 4 is given in Table 8. Note that since associations in UML are always between types, the OWL property always has domain and range specified. If the association name occurs more than once in the same model, it must be disambiguated in the OWL translation, for example by concatenating the member names to the association name.

Table 8

Association	Member 1 Property Type	Member 2 Property Type	OWL equivalent
enrolled	Course	Student	<owl:ObjectProperty rdf:ID="enrolled"> <rdfs:domain rdf:resource="Course"/> <rdfs:range rdf:resource="Student"/>

			</owl:ObjectProperty>
--	--	--	-----------------------

Both languages support the **subclass** relationship (OWL `rdfs:subClassOf`, UML generalization). Both also support **subproperties** (UML generalization of association). UML defines generalization at the supertype *classifier*, while in OWL subtype and subproperty are separately but identically defined.

The translation from UML to OWL is straightforward. If $\langle S, G \rangle$ is an M1 instance of the UML M2 association *generalization* (S is a subclassifier of G), then if both S and G are classes and TS, TG are respectively the types of the identifying owned property of S, G respectively, the OWL equivalent is the addition of the clause

`<rdfs:subClassOf rdf:resource="TG"/>`

to the definition of the OWL class TS . Similarly if S and G are both associations, the owl equivalent is the addition of the clause

`<rdfs:subPropertyOf rdf:resource="G"/>`

to the definition of the OWL object property S .

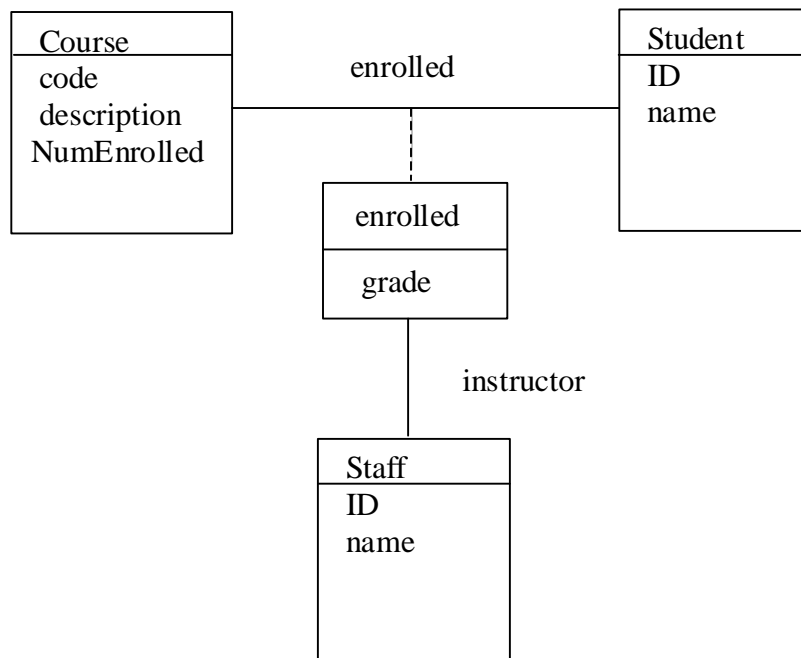


Figure 3. M1 model with association class

An association in UML can be N-ary. It can have a non-navigable end (*ownedEnd*). It can also be a class (*association class*), so can participate in further associations. In OWL DL, classes and properties are disjoint, but in OWL Full they are overlapping. However, there is limited syntactic mechanism in the documents so far published to support this overlap. There is an advantage in translating these more complex associations to structures supported by OWL DL. In any case, the translations proposed are not normative, so those

responsible for a particular application can use more powerful features of OWL if there is an advantage to doing so.

Our proposal takes advantage of the fact that an N-ary relation among types $T_1 \dots T_N$ is formally equivalent to a set R of identifiers together with N projection functions P_1, \dots, P_N , where $P_i:R \rightarrow T_i$. Thereby N-ary UML associations are translated to OWL classes with bundles of binary functional properties.

The model of Figure 3 is represented in table form in Table 9.

Table 9

Association	End	Type
enrolled	1	Course
	2	Student
	3	Grade
	4	enrolled
instructor	1	enrolled
	2	Staff

Instructor is translated into an OWL property in the same way as shown in Table 8. However, *enrolled* would be translated into the following OWL statement:

```

<owl:Class rdf:ID="enrolled" / >
<owl:FunctionalProperty rdf:ID="enrolledCourse">
  <rdfs:domain rdf:resource="enrolled"/>
  <rdfs: range rdf:resource="Course"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledStudent">
  <rdfs:domain rdf:resource="enrolled"/>
  <rdfs: range rdf:resource="Student"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledGrade">
  <rdfs:domain rdf:resource="enrolled"/>
  <rdfs: range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty >
<owl:FunctionalProperty rdf:ID="enrolledenrolled">
  <rdfs:domain rdf:resource="enrolled"/>
  <rdfs: range rdf:resource="enrolled"/>
</owl:FunctionalProperty >

```

2.3 More advanced concepts

There are a number of more advanced concepts in both UML and OWL. In the cases where the UML concept occurs in OWL, the translation is often quite straightforward, so will not always be shown.

Both languages support a module structure, called **package** in UML and **ontology** in OWL. The translation of package to ontology is straightforward.

Both UML and OWL support a fixed defined extent for a class (OWL **oneOf**, UML **enumeration**).

UML has the option for binary associations to have distinguished ends which can be **navigable** or **non-navigable**. A navigable property is one which is owned by a class, while a non-navigable is not (an integer, say). OWL properties always are binary and have distinguished ends called **domain** and **range**. A UML binary association with one navigable end and one non-navigable end will be translated into a property whose domain is the navigable end. A UML binary association with two navigable ends will be translated into a pair of OWL properties, where one is **inverseOf** the other.

A key difference is that in OWL a property is defined by default as having range and domain both *Thing*. A given property therefore can in principle apply to any class. So a property name has global scope and is the same property wherever it appears. In UML the scope of a property is limited to the subclasses of the class on which it is defined. A UML association name can be duplicated in a given diagram, with each occurrence having a different semantics.

An OWL individual can therefore be outside the system in a UML model. UML has a facility **dynamic classification** which allows an instance of one class to be changed into an instance of another, which captures some of the features of Individual, but an object must always be an instance of some (non-universal) class.

Both languages allow a class to be a subclass of more than one class (**multiple inheritance**). Both allow subclasses of a class to be declared **disjoint**. UML allows a collection of subclasses to be declared to **cover** a superclass, that is to say every instance of the superclass is an instance of at least one of the subclasses. The corresponding OWL construct is the declare the superclass to be the union of the subclasses, using the construct **unionOf**. (Note that the OWL construct *unionOf* applies to other RDF resources than classes, so this is a restricted use.)

UML has a strict separation of metalevels, so that the population of M1 classes is distinct from the population of M0 instances. OWL Full permits classes to be instances of other classes.

In OWL, a property when applied to a class can be constrained by cardinality restrictions on the domain giving the minimum (**minCardinality**) and maximum (**maxCardinality**) number of instances which can participate in the relation. In addition, an OWL property can be globally declared as functional (**functionalProperty**) or inverse functional (**inverseFunctional**). A functional property has a maximum cardinality of 1 on its range, while an inverse functional property has a maximum cardinality of 1 on its domain. In UML an association can have minimum and maximum cardinalities (**multiplicity**) specified for any of its ends. OWL allows individual-valued properties (*objectProperty*) to be declared in pairs, one the inverse of the other.

So if a binary UML association has a multiplicity on a navigable end, the corresponding OWL property will have the same multiplicity. If a binary UML association has a multiplicity on its both ends, then the corresponding OWL property will be an inverse pair, each having one of the multiplicity declarations.

For an N-ary UML association, any multiplicity associated with one of its UML properties will apply to the OWL property translating the corresponding projection.

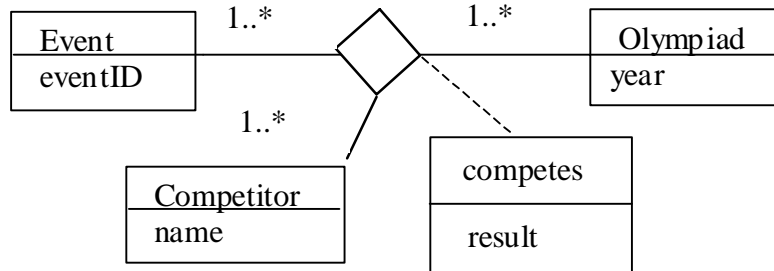


Figure 4. Example N-ary association with multiplicity

The N-ary association in Figure 4 would be translated as below, assuming that the attribute *result* has multiplicity 1..1. Note that there are several alternative OWL syntaxes. This particular version has inline restrictions with no XML Entity Declarations. It is the simplest, and since UML associations are distinct this version reflects UML well. Should a particular application wish to use a model translated from UML as a base for further development in OWL, an appropriate variant may be used.

```

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf=" http://www.w3.org/1999/02/22-rdf-syntax-ns
  <http://www.w3.org/1999/02/22-rdf-syntax-ns> #"
  xmlns:rdfs=" http://www.w3.org/2000/01/rdf-schema
  <http://www.w3.org/2000/01/rdf-schema> #"
  xmlns:owl=" http://www.w3.org/2002/07/owl <http://www.w3.org/2002/07/owl>
  #" xmlns:xsd=" http://www.w3.org/2001/XMLSchema
  <http://www.w3.org/2001/XMLSchema> #"
>
<owl:Class rdf:ID="competes">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="competesEvent"/>
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="competesCompetitor"/>
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#competesOlympiad"/>
      <owl:minCardinality
        rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:subClassOf>

```

```

    <owl:Restriction>
      <owl:onProperty rdf:resource="#competesResult"/>
        <owl:minCardinality
          rdf:datatype="xsd:nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:subClassOf>
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#competesResult"/>
        <owl:maxCardinality
          rdf:datatype="xsd:nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
<owl:FunctionalProperty rdf:ID="competesEvent">
  <rdfs:domain rdf:resource="#competes"/>
  <rdfs:range rdf:resource="#Event"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesCompetitor">
  <rdfs:domain rdf:resource="#competes"/>
  <rdfs:range rdf:resource="#Competitor"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesOlympiad">
  <rdfs:domain rdf:resource="#competes"/>
  <rdfs:range rdf:resource="#Olympiad"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="competesResult">
  <rdfs:domain rdf:resource="#competes"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
<http://www.w3.org/2001/XMLSchema#string> >
</owl:FunctionalProperty>
</rdf:RDF>

```

The difference in scope of property names leads to a difference in the use of cardinality restrictions. In UML an association with its multiplicity is generally declared only once, whereas an OWL property can have different (compatible) cardinalities when applied to different classes.

Note that the class might be the domain of a property for which the individual might not have a value. This can happen if the mincardinality of the domain of the property is 0 (or maxcardinality < mincardinality)³, in which case the property is optional (or partial) for that class. The same can happen in UML. An instance of a class is constrained to participate only in properties which are mandatory, minimum cardinality >0. So an instance can lack optional properties.

However, even if the property is mandatory (mincardinality > 0 and maxcardinality >= mincardinality), there may not be definite values for the property. Consider a class (K) for which a property (P) is mandatory. In this case, the individual (I) must satisfy the predicate

³ This is a somewhat strange construct. It is syntactically correct in OWL and has the semantics that the property has no instances. It can occur where multiple autonomous ontologies are merged, for example.

[M]: I instance of K \rightarrow exists X such that $P(I) = X$.

It is not required in OWL that there be a constant C such that $X = C$. All horses have color, but we may not know what color a particular horse has.

In UML, there is a strict separation between the M1 and M0 levels. At the M1 level, that an association is mandatory (minimum cardinality greater than 0) is exactly the predicate [M]. Any difference between UML and OWL must come from the treatment of the model of the M1 theory at the M0 level. In practice, M0 models in UML applications tend to be Herbrand models implemented by something like an SQL database manager. For these cases, if we know a horse has a color, then we know what color it has.

But UML does not mandate M0 models to be Herbrand models. In particular SQL-92 supports the Null value construct, which has multiple interpretations, including “value exists but is not known”. Some years ago, CJ Date proposed a zoo of nulls with specific meanings, including “value exists but is not known”, and there have been proposals by Ray Reiter and others for databases with either existentially quantified variables in the data or which reason with the M1 theory for existentially quantified queries. It is possible for a particular application to introduce a special constant “unknown” into a class, which is treated specially by the programs. UML does not forbid an implementation of a class model in one of these ways. So there is no difference in principle between UML and OWL for properties which are declared to have minCardinality greater than 0 (and maxCardinality \geq minCardinality) for a class.

Note that a consequence of this possible indeterminacy, it may not be possible to compute a transitive closure for a property across several ontologies, even if they share individuals.

An OWL property can have its range restricted when applied to a particular class, either that the range is limited to a class (subclass of *range* if declared) (**allValuesFrom**) or that the range must intersect a class (**someValuesFrom**).

OWL allows properties to be declared symmetric (**SymmetricProperty**) or transitive (**TransitiveProperty**). In both cases the domain and range must be type compatible.

OWL permits declaration of a property whose value is the same for all instances of a class, so the property value is in effect attached to the class (OWL DL property declared as allValuesFrom a singleton set for that class). OWL full allows properties to be directly assigned to classes without special machinery. If class A is an instance of class B, then a property P whose domain includes B will designate a value P(A) which applies to the class A so is common to all instances of A.

UML allows a property to be **derived** from other model constructs, for example a composition of associations or from a generalization.

Two different objects modeled in UML may have dependencies which are not represented by UML named (model) elements, so that a change in one (the supplier) requiring a change in the other (the client) will not be signaled by for example association links. Two such objects may be declared **dependent**. There are a number of subclasses of dependency, including abstraction, usage, permission, realization and substitution. OWL does not have a comparable feature, but RDF, the parent of OWL, permits an RDF:property relation between very general elements classified by RDFS:Class.

Therefore, a dependency relationship between a supplier and client UML model element will be translated to a reserved name RDF:Property relation whose domain and range are both RDF:Class. Population of the property will include the individuals which are the target of the translation of the supplier and client named elements.

2.4 Summary of more or less common features

This section has described features of UML and OWL which are in most respects similar. Table 10 summarizes the features of UML in this feature space, giving the equivalent OWL features. UML features are grouped in clusters which translate to a single OWL feature or a cluster of related OWL features. The column *Package* shows the section of the UML Superstructure document [2] where the relevant features are documented.

Table 10

UML features	Package	OWL features	Comment
class, property ownedAttribute ,type ⁴	7.11 Classes 7.8 Classifiers 7.4 Multiplicities	class	
instance	7.7 Instances	individual	OWL individual independent of class
ownedAttribute, binary association	7.11 Classes	property	OWL property can be global
subclass, generalization	7.11 Classes 7.8 Classifiers	subclass subproperty	
N-ary association, association class	7.11 Classes 7.16 Association Classes	class, property	
enumeration	7.12 Datatypes	oneOf	
navigable, non- navigable	7.2 Root	domain, range	
disjoint, cover	7.17 Powersets	disjointWith, unionOf	
multiplicity	7.4 Multiplicities	minCardinality maxCardinality inverseOf	OWL cardinality declared for each class
derived	7.11 Classes	no equivalent	
package	7.13 Packages	ontology	
dependency	7.14 Dependencies	reserved name RDF:properties	

⁴ This cell summarizes the relationship between UML class and OWL class mediated by property, ownedAttribute and type. It does not signify that the latter three are themselves translated to OWL class.

All of the UML features considered in the scope of the ODM have more-or-less satisfactory OWL equivalents. Some OWL features in this feature space have no UML equivalent, so are omitted from Table 10. They are summarized in Table 11.

Table 11

OWL features with no UML equivalent
Thing, global properties, autonomous individual
class-specific cardinality redefinition ⁵
allValuesFrom, someValuesFrom
SymmetricProperty, TransitiveProperty
Classes as instances

3. OWL but not UML

3.1 Predicate definition language

OWL permits a subclass to be declared using `subclassOf` or to be inferred from the definition of a class in terms of other classes. It also permits a class to be defined as the set of individuals which satisfy a restriction expression. These expressions can be a boolean combination of other classes (**intersectionOf**, **unionOf**, **complementOf**), or property value restriction on properties (requirement that a given property have a certain value – **hasValue**). **EquivalentClass** applied to restriction expressions can be used to define classes based on property restrictions.

For example, the class definition⁶

```
<owl:Class rdf:ID="TexasThings">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#locatedIn" />
      <owl:allValuesFrom rdf:resource="#TexasRegion" />
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Defines the class *TexasThings* as a subclass of the domain of the property *locatedIn*. These individuals are precisely those for which the range of *locatedIn* is in the class *TexasRegion*. Given that we know an individual to be an instance of *TexasThings*, we can infer that it has the property *locatedIn*, and all of the values of *locatedIn* associated with it are instances of *TexasRegion*. Conversely, if we have an individual which has the property *locatedIn* and all of the values of *locatedIn* associated with that individual are in *TexasRegion*, we can infer that the individual is an instance of *TexasThings*.

⁵ UML permits specializations of associations, but the current version of the superstructure specification is silent on whether multiplicities can be redefined

⁶ OWL Web Ontology Language Guide <http://www.w3.org/TR/2003/PR-owl-guide-20031215/> section 3.4.1

Because it is possible to infer from the properties of an individual that it is a member of a given class, we can think of the complex classes and property restrictions as a sort of predicate definition language.

UML provides but does not mandate the predicate definition language OCL.

OCL and SCL (Simple Common Logic) are two predicate definition languages which are relevant to the ODM. Both are more expressive than the complex class and property restriction expressions of OWL Full. There are also other predicate definition languages of varying expressive powers which particular applications might wish to use.

The ODM will not mandate any particular predicate definition language, but will provide a place for a package enabling the predicate definition language of choice for an application.

3.2 Names

A common assumption in computing applications is that within a namespace the same name always refers to the same object, and that different names always refer to different objects (the **unique name assumption**). As a consequence, given a set of names, one can count the names and infer that the names refer to that number of objects.

Names in OWL do not by default satisfy the unique name assumption. The same name always refers to the same object, but a given object may be referred to by several different names. Therefore counting a set of names does not warrant the inference that the set refers to that number of objects. Names, however, are conceptually constants, not variables.

OWL provides features to discipline names. The unique name assumption can be declared to apply to a set of names (**allDifferent**). One name can be declared to refer to the same object as another (**sameAs**). One name can be declared to refer to something different from that referred to by any of a set of names (**differentFrom**).

Classes and properties are by default different, but two classes or two properties can be stated to be equivalent (**equivalentClass**, **equivalentProperty**).

UML at the M1 level has names only for classes and properties. Although a UML class may be defined to contain a definite collection of names, names are the province of M0. Applications modeled in UML are frequently implemented using systems like SQL which default the unique name assumption, but this is not mandated. UML places no constraints on names at the M0 level.

In particular, it is permitted for applications modeled in UML to be implemented at the M0 level using names which are variables. Note that the UML constraint language OCL uses variables. OWL does not support variables at all.

It is proposed that the ODM adopt the OWL naming system.

3.3 Other OWL developments

There are a number of developments related to OWL which are not yet finalized, including SWRL Semantic Web Rule Language and OWL services. These are considered

out of scope for the ODM. A translation of an out-of-scope model element will be to a comment in the OWL target.

4. In UML but not OWL

4.1 Behavioral features

UML allows the specification of behavioral features, which are essentially programs. One use of behavioral features is to calculate property values. This use has already been considered in the properties section above (*derived properties*). Other programs would presumably have side effects. Facilities of UML supporting programs include **operations**, which are method names; **responsibilities**, which specify which class is responsible for what action; **static operations**, which are operations attached to a class like static attributes; **interface classes**, which specify interfaces to operations; **abstract classes**, whose operations are specified in subclasses; **qualified associations**, which are programming language data structures; and **active classes**, which are classes each instance of which controls its own thread of execution control.

It is proposed that the ODM omit behavioral features of UML.

4.2 Complex objects

UML supports various flavors of the part-of relationship between classes. In general, a class (of parts) can have a part-of relationship with more than one class (of wholes). One flavor (**composition**) specifies that every instance of a given class (of parts) can be a part of at most one whole. Another (**aggregation**) specifies that instances of parts can be shared among instances of wholes.

Composite structures are runtime instances of classes collaborating via connections. They are used to hierarchically decompose a class into its internal structure which allows a complex objects to be broken down into parts. These diagrams extend the capabilities of class diagrams, which do not specify how internal parts are organized within a containing class and have no direct means of specifying how interfaces of internal parts interact with its environment.

Ports and Connectors model how internal instances are to be organized. Ports define an interaction point between a class and its environment or a class and its contents. They allow you to group the required and provided interfaces into logical interactions that a component has with the outside world. **Collaboration** provides constructs for modeling roles played by connectors.

Comparing complex objects can be problematic, because often a whole object is considered to remain “the same” even though some of its parts might change. UML supports **reference objects**, which are the same if they have the same name regardless of content, and **value objects**, which need to have the same content to be the same.

Although not strictly part of the complex object feature set, the feature **template** (parameterized class) is most useful where the parameterized class is complex. One could for example define a multimedia object class for movies, and use it as a template for a collection of classes of genres of movie, or a complex object giving the results of the

instrumentation on a fusion reactor which would be a template for classes containing the results of experiments with different objectives.

Although it is recognized that there is a need for facilities to model mereotopological relationships in ontologies, there does not seem to be sufficient agreement on the scope and semantics of existing models for inclusion of specific mereotopological modeling features into the ODM at this stage.

These modeling elements will be translated to properties or classes as ownedAttributes or association ends. The target elements will be annotated with appropriate comments.

4.3 Access control

UML permits a property to be designated **read-only**. It also allows classes to have **public** and **private** elements.

It is proposed that the ODM omit access control features.

4.4 Keywords

UML has keywords which are used to extend the functionality of the basic diagrams. They also reduce the amount of symbols to remember by replacing them with standard arrows and boxes and attaching a <<keyword>> between guillemets. A common feature that uses this is <<interfaces>>.

It is proposed that the ODM omit this feature.

5. References

[1] OWL Web Ontology Language Overview , W3C Proposed Recommendation 15 December 2003, <http://www.w3.org/TR/2003/PR-owl-features-20031215/>

[2]

http://www.omg.org/techprocess/meetings/schedule/UML_2.0_Superstructure_FT.html