# Alternative Formalizations of Aggregations and Associations in UML

Jeffrey Smith[1], Mieczyslaw Kokar[2] and Kenneth Baclawski[2]

[1] Sanders, a Lockheed Martin Company
[2] Northeastern University

**Abstract.** The UML specification contributors have made great strides in defining semi-formal semantics, with a combination of meta-language, constraint specification and text, in the UML Semantics Guide. Improvements in these semi-formal UML semantic descriptions are needed to convey a rigorous semantic representation and provide tool support to verify UML Diagrams against an unambiguous specification of UML semantics. We describe a step towards improvement of UML semantics by 1) formalizing the (meta)objects in the UML Semantics Guide using the Slang formal methods language and 2) translating UML Diagrams into a Slang form compatible with our UML formalization. We focus on various specifications of aggregation and association formalizations, to distinguish their semantic differences, with a rationale and partial formalization for recommended and alternative association/aggregation formalizations. We then give a specific example of the association/aggregation formal translation and specification process. In this example, we verify that an automatically generated Slang form of a UML Diagram is consistent with our formalization of UML Semantics.

**Keywords -** UML specification, formalization and translation, formal methods.

## 1 UML Formalization Process

The UML specification contributors have defined semi-formal semantics, with a combination of meta-language, constraint specification and text, in the UML Semantics Guide. Improvements in these semi-formal UML semantic descriptions are needed to 1) convey a more rigorous semantic representation and 2) provide tool support to enforce and verify UML Diagrams against an unambiguous specification of object-oriented semantics. This paper describes a step towards improvement of UML semantics.

Our UML semantic formalization process is shown in Figure 1. Although, the Slang formal methods language [W+98] is used to give formal specification language examples and references, the concepts described in this paper can be attributed to any algebraic/category theory based formal language. Since UML is described in UML, Transition 1 describes the formalization of UML

(meta)objects in Slang, with a rationale for each formalization rule and a description of formalization rule alternatives. Transition 2 shows the tool support needed to automatically translate UML applications (described as the UML Graphical Domain) to Slang. In Transition 3, we check that instances of the Slang form of the UML Graphical Domain are compatible with our Slang representation of abstract UML theory, viz. the UML Formal Semantics in Figure 1. Transition 4 shows the check of UML semantics and constraints performed within a Computer-Aided Software Engineering (CASE) tool. We will show that there is an satisfaction relation, that follows from our formalization process, between the UML Graphical Domain and Formal Semantics.

An entire UML formalization is too lengthy a topic for this paper. We will

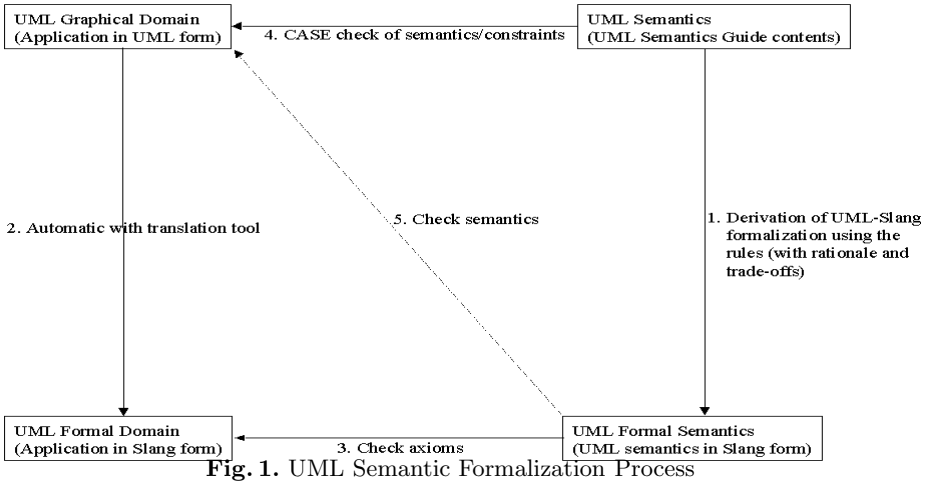


**Fig. 1.** UML Semantic Formalization Process

describe a formalization of aggregation and association, not only to show an example of our process, but also to contribute a semantic distinction between these two relationships since "there is no single accepted definition of the difference between aggregation and association used by all methodologists" [MF97]. Figure 2 shows the subset of the UML Semantics Guide Core Package-Relationships Diagram [UMLSEM], that we will address in this paper.

Our approach has been to formalize UML in category theory in a manner that closely follows the UML Semantics Guide. Alternative approaches generally formalize by mapping concepts in UML to a priori constructs such as "has-a" and "part-of" that don't necessarily follow UML very closely for two reasons: (1) UML treats aggregation as a property of one of the association ends of an association rather than as a separate modeling primitive. (2) There are several notions of UML aggregation, but there is just one "part-of" modeling primitive. The following sections will trace each of the steps of Figure 1 for a UML appli-

cation, demonstrating how we use a Slang form of UML semantics to verify that UML Diagrams are consistent with UML specifications.

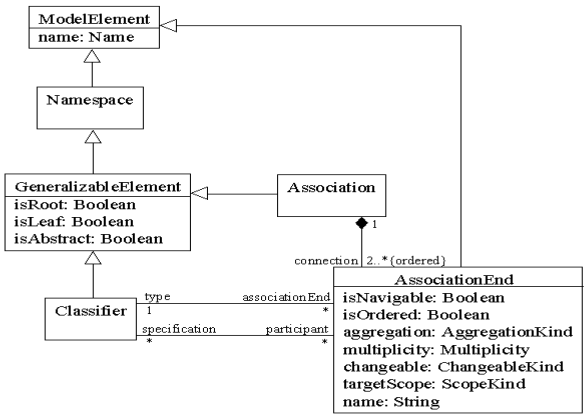


**Fig. 2.** Aggregation and Association Portion of UML Core Package-Relationships Diagram

## 2 UML/Slang Formalization

This section describes how to formalize Diagrams from the UML Semantics Guide in an algebraic/category theory based specification language. UML to formal specification translation is broken into a modular set of rules to break this large problem into chunks.

### 2.1 Specifications in Slang

The following, from [Spe98], gives a brief overview of specifications expressed in Slang. Specifications are the fundamental objects in Slang. A specification is viewed as a presentation or description of a theory. A specification is a finite expression that describes a potentially infinite set of strings of symbols that are within the language of a theory and a subset of this set of strings that are valid within a theory. Legal sentences are described by *signatures*, providing a presentation of sorts and operations that the theory deals with. Signatures are made up of *sorts* (a declaration of the classes of objects in the specification), *ops* (ops are short for operations - a declaration of named constants that denote objects, functions and predicates of specified sorts) and *sort axioms* (an assertion of the equivalence between a primitive sort and a constructed sort).

Specifications form a category called **Spec**. The objects in this category are specifications (Specs) and arrows are such morphisms that map sorts to sorts, operations to operations, and axioms to theorems. Specifications are either given as basic specifications of these sorts, ops, axioms and theorems, or built with *translate, import* or *colimit* specification-building operations. *Translate* creates a copy of a specification, sometimes renaming some components. *Import* enriches a specification with new sorts, operations, axioms and theorems - similar to a programming language *include*. A *colimit* is used to combine specifications, taking a diagram of specifications as an input and yielding a specification that contains all the elements of the specifications in the diagram.

### 2.2 Background Formalization Rules

We begin with some background formalization rules used by the subsequent formalizations of associations and aggregations.

**Object - Spec Rule.** Every Model Element in UML, specified in the UML Semantics Guide, translates to a Spec containing a sort, both having the same name as the Model Element. Import a CLASS Spec if the Model Element represents a class and an OBJECT Spec if the Model Element represents an object. To make a lexical distinction between a Spec and a sort name, the sort name will begin with a capital letter and then will use lower case for the remaining characters of the name, while the Spec name will use all upper case letters.

There are only two choices for an object in a specification language supporting category theory: a Diagram or a Spec. Of these two choices, only Spec supports the Import declarations we need to support a modular translation. The Spec choice also permits more degrees of freedom than Diagram because one can also associate operations (and other constructs not possible with Diagrams) with a Spec. Specs are the fundamental objects in Slang. They are used to describe domains, data structures and programs, at multiple levels.

**OCL Constraints to Op/Axiom Rule.** For each OCL constraint, add an associated op in the Spec corresponding to the UML object that contains this OCL constraint. Specify the constraint in an axiom associated with the op.

A constraint defines a relation. In Slang, relations are defined as ops with Boolean as their domain. For this reason, in order to specify an OCL constraint in Slang, we first have to specify a boolean op and then specify the constraint as either an axiom or definition associated with this op. We specify the constraint with axioms because, unlike definitions, they restrict the Spec they are defined in. This follows since the objective of OCL is to restrict operations on objects. Definitions are only used for naming purposes.

**Attribute Rule.** For every class/object that includes an attribute, distinguish the class/object Spec by adding the suffix "-BASE" to the class/object name

and the sort of the same class/object name (identified in the ModelElement-Spec translation rule), as well as to everywhere the sort of the same class/object name is referenced in the class/object Spec (e.g. those referenced in the OCL Constraints to Op/Axiom translation rule). Translate each attribute to a separate Spec, whose name is prefixed with the attribute name and postfixed with "-ATTRIBUTE" (note this attribute Spec also has a sort of the same name). Each attribute Spec includes an op, where the domain of the op is the attribute name, prefixed by "this_" and the range of the op is the sort (type) of the attribute. Translate each class/object, that included the attribute, to a separate colimit Spec, whose name is the name of the class/object. This colimit Spec maps each attribute's sort, identified in the separate attribute Specs, to the class/object sort and each op, prefixed by "this_" in the attribute Spec, to the attribute name.

An attribute takes an object name and returns information about the object. The reason for a separate attribute Spec is for consistency with the ModelElement-Spec translation rule, since an attribute is also a model element. Another reason is to handle attributes in a similar fashion as associations and generalizations. All three cases involve circular references, since the attribute, association and generalization model elements also use attributes, associations and generalizations in their specifications. Associations and generalizations were modeled as separate Specs, so it make sense to handle attribute specification similarly. The reasoning behind the renaming of classes/object Specs and sorts that include attributes is so that the colimit, unifying the attributes and classes/objects, can have a Spec name and sort of the same class/object name (just as Specs of class/objects without attributes). This is also important since the aggregation, association and generalization Specs expect to use these sorts of the same class/object name, whether the class/object has attributes or not.

Note that although an op (in the attribute Spec) can only map to a single sort, one can formalize multiple valued attributes and attributes that are not defined everywhere by mapping to a structured sort.

**Import for Attribute - Op Rule.** For each attribute (sort) that is referenced by a Spec, but not defined in the Spec, import the Spec in which this attribute is defined using the Attribute - Op Rule.

Prior OMT formalization research [DeL96,BC95] identified a similar rule stating, "If D references a separate class specification, then the specification for D is included into the class specification." Our assumption is that the included class specification is already defined algebraically and includes a specification of a sort of the same name as the class specification. Any possibility of infinite recursion, resulting from Specs mutually referring to each other, is eliminated with type checking. The purpose of this rule is to support attribute definitions that use externally defined classes (needed for modularity) and is necessary to type check a formal language specification.

### 2.3 UML Association Formalization Rule

**Association - Association Instance Spec Rule.** Translate each association to a separate instance of the ASSOCIATION Spec which imports pairs of ASSOCIATIONEND Specs, filling in each of the association end constraints in the association instance Spec.

The motivation to use this translation rule is to closely resemble the UML Semantics Guide, where an association is a set of tuples relating two classifiers. An association consists of at least two association ends, each of which represents a connection of an association to a classifier. This translation rule uses the UML intended structure. As a result, associations and association ends are also translated to Specs. This seems apt since we're translating objects to Specs and Associations and AssociationEnds are meta-objects in Figure 2.

*Formalization.* The following describes the ASSOCIATIONEND, CONNECTION and ASSOCIATION Specs that will be used in an example that follows.

```
spec ASSOCIATIONEND is
  import MODELELEMENT, MODELELEMENT-SET
  sorts AssociationEnd, Aggregate, Changeable
  sort-axiom Changeable = String | chgb?
  op chgb? : String -> Boolean
  axiom chgb?(x) <=> (x = "none" or x =  "frozen" or x = "addOnly")
   sort-axiom Aggregate = String | aggr?
  op aggr?: String -> Boolean
  axiom aggr?(x) <=> (x = "none" or x = "aggregate" or x = "composite")
  op isNavigable: AssociationEnd -> Boolean
  op isOrdered: AssociationEnd -> Boolean
  op name: AssociationEnd -> String
  op aggregation: AssociationEnd -> Aggregate
  op multiplicity: AssociationEnd -> Nat
  op changeable: AssociationEnd -> Changeable
  op type: AssociationEnd -> String % string contains Classifier AE is connected to
% op targetScope: will be resolved with resolution of rules to translate
% classifiers and instances. Qualifier, specification associations are left unspecified.
  op firstend: AssociationEnd -> Name
  op otherend: AssociationEnd -> Name
 end-spec

spec CONNECTION is
% CONNECTION is pair of association ends
  import ASSOCIATIONEND
  sorts Connection
  op make-connection: AssociationEnd, AssociationEnd -> Connection
  op first:  Connection -> AssociationEnd
  op second: Connection -> AssociationEnd
  axiom first(make-connection(d, e)) = d
  axiom second(make-connection(d, e)) = e
  constructors {make-connection} construct Connection
  theorem p = make-connection(first(p), second(p))
 end-spec

spec ASSOCIATION is
  import GENERALIZABLEELEMENT, CONNECTION
  sort Association
  sort-axiom ModelElement = Association
  op name: Association -> Name % OCL 1 - uniqueness of this name within the
% enclosing namespace for sort Name is enforced within the NAMESPACE specification
  op assocOCL2: Connection -> Boolean % OCL 2 - at most 1 association end be an association or aggregate
  axiom assocOCL2(A) <=>
   (relax(aggr?)(aggregation(first(A))) = "aggregation") or
     (relax(aggr?)(aggregation(first(A))) = "composite")=>(relax(aggr?)(aggregation(second(A))) = "none")
% OCL 3 - not possible yet because an association can only have 2 ends in our restricted UML spec
% OCL 4 - not needed because the connected classifier of association ends will be
```

```
% included in the namespace of association by construction
  theorem commutivity-of-symmetric-association is ((first(a) = second(b) & first(b) = second (a)) => a = b)
 end-spec

spec ASSOCIATION-COLIMIT is
  translate colimit of diagram
  nodes TRIV, SET, CONNECTION
  arcs
    TRIV -> SET : {},
    TRIV -> CONNECTION : {E -> CONNECTION}
  end-diagram
    by {Set -> Association}
```

**Aggregation - Aggregation Instance Spec Rule.** Treat aggregation as an association translation, labeling the association end corresponding to the aggregate end (the side with the hollow or filled in diamond) with the type of aggregation.

An UML aggregation and a colimit model a "part-of" relationship. If B is a part of A, then B belongs to A. This is contrasted with an association, or "is-a" relationship, where if B is associated with A, then the B-A relationship belongs to the superspec that imports this relationship. Aggregation is a special type of association. Note, there is no aggregation meta-object in the UML semantics specification. Aggregation (and type of aggregation) is an attribute of the association ends that make up an association. By selecting an association translation rule first, we have a singular choice when choosing this translation rule. In our Association translation Rule, and in UML, an aggregation is an association where one of the ends of an association connection is marked with the type of aggregation. It is marked as *aggregate* if the other end, or part, may be contained in other aggregates. It is marked as *composite* if the other end may not be part of any other composite.

## 3 UML Graphical to Formal Domain Translation

This UML Graphical to Formal Domain translation was depicted as Transition 2 in Figure 1, where we provide support to translate a UML application to a Slang form of the same application, following the previously defined translation rules. In this association/aggregation example (Figure 3), a *Lecture* is a collection of *Student*, ordered by ID. There is a one-to-one association with a *Course*, depending on the Lecture level. The translation of this UML Diagram to Slang, according to the prior semantic formalization rules, looks like the following.

```
spec LECTURE is
  import CLASS, PRESENTATION
  sort Lecture
  axiom name(Lecture) = "Lecture"
end-spec

spec COURSE is
  import CLASS
  sort Course
  axiom name(Course) = "Course"
end-spec
```

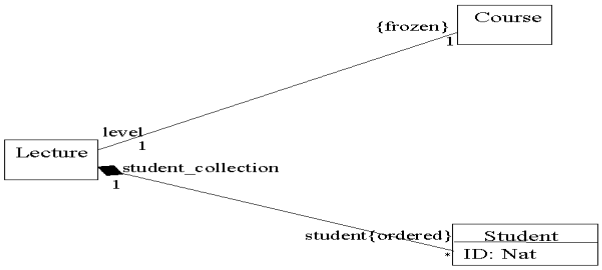


**Fig. 3.** UML Association and Aggregation Translation Example

```
spec STUDENT-BASE is
  import CLASS
  sort Student-base
  axiom name(Student-base) = "Student"
end-spec

spec ID-ATTRIBUTE is
  sort ID-attribute
  op this_ID: ID-attribute -> Nat
end-spec

spec Student is
  translate colimit of diagram
  nodes ONE-SORT, STUDENT-BASE, STUDENT-ATTRIBUTE
arcs
  ONE-SORT -> STUDENT-BASE: {x -> Student-base},
  ONE-SORT -> STUDENT-ATTRIBUTE: {x -> Student-attribute}
end-diagram
  by {Student-attribute -> Student-base, this_Student -> Student}

spec LECTURE-STUDENT-AGGREGATION is
  import ASSOCIATION
  sort Lecture-Student-Aggregation
  axiom multiplicity(first(Lecture-Student-Aggregation)) = 1..1
  axiom name(first(Lecture-Student-Aggregation)) = "student_collection"
  axiom relax(aggr?)(aggregation(first(Lecture-Student-Aggregation))) = "aggregate"
  axiom firstend(first(Lecture-Student-Aggregation)) = Lecture
  axiom name(second(Lecture-Student-Aggregation)) = "student"
  axiom otherend(second(Lecture-Student-Aggregation)) = Student
  axiom isOrdered(second(Lecture-Student-Aggregation)) = true
end-spec

spec LECTURE-COURSE-ASSOCIATION is
  import ASSOCIATION
  sort Lecture-Course-Association
  axiom multiplicity(first(Lecture-Course-Association)) = 1..1
  axiom name(first(Lecture-Course-Association)) = "level"
  axiom firstend(first(Lecture-Course-Association)) = Lecture
  axiom multiplicity(second(Lecture-Course-Association)) = 1..1
  axiom otherend(second(Lecture-Course-Association)) = Course
  axiom relax(chgb?)(changeable(second(Lecture-Course-Association))) = "frozen"
end-spec

{I'm thinking about adding an association colimit here}
```

## 4 Check UML Formal Semantics

{ this part needs to be amended/replaced with our current thinking }

In Transition 3 in Figure 1, we are trying to verify that the Slang form of our application is an instance of the UML abstract theory described in the UML Formal Semantics. This verification process checks whether the application satisfies the axioms in the UML Formal Semantics.

Continuing with the example begun in the last section, let's focus on the LECTURE-COURSE-ASSOCIATION Spec. If the association ends were reversed in a new Spec, where the first end was connected to *Course* and the second end were connected to *Lecture*, then we can prove that the meaning of the two symmetric associations are the same by virtue of the *commutivity-of-symmetric-association* theorem in the ASSOCIATION Spec: *theorem commutivity-of-symmetric-association is ((first(a) = second(b) & first(b) = second (a)) → a = b)*

Another example is the check of the satisfaction of the multiplicity constraint, where we check whether the multiplicity of the aggregation of the first end of the association between a Lecture and a Student, which is **1**, is compatible with the sort of multiplicity allowed for this attribute. Note that an association (or aggregation) is a pair (Spec CONNECTION) of ASSOCIATIONEND Specs. Associations include sets of these connection pairs. The range of the *multiplicity* op in the ASSOCIATIONEND Spec is of type Nat, which is compatible with 1.

Similarly, the *name* of the *firstend* is **"student_collection"**, is compatible with the range of the *name* op of the ASSOCIATIONEND Spec, which is of type string. The *aggregation* of the *firstend* is **"composite"**. The associated *aggregation* axiom in the ASSOCIATIONEND Spec specifies that aggregation must not only be of sort string, but of subsort **"none"**, **"aggregate"** or **"composite"**. This check that a UML Formal Domain is an instance of the UML Formal Semantics may be performed for any application.

## 5 Check UML Graphical Domain

Typically, one is dependent on a CASE tool to check that a UML Diagram doesn't violate UML semantics as part of the Diagram construction process (as in Transition 4 of Figure 1). The problem is that CASE tools don't enforce a complete set of UML Semantics Guide specifications and constraints. For instance, there is typically no difference between a one-to-one association or aggregation relation in a CASE model file. The satisfaction relation, in Transition 5 of Figure 1, is implicit in our formalization process since there is an satisfaction relation between the UML 1) Graphical and Formal Domain, 2) Formal Domain and Formal Semantics and 3) Semantics and Graphical Domain.

## 6 Future Research

In addition to the translation from UML to Slang, Smith and DeLoach have built a translator that automatically translates from UML directly to O-Slang.

O-Slang is an object-oriented form of Slang that DeLoach had built in [DeL96]. This translation tool includes both static class and behavior (state) translation.

The translation rules given in this paper are a subset of a larger research effort which currently includes only static class formalizations. Other portions of this research will be addressed in subsequent papers, due to the size of each formalization. This research effort will implement axioms to capture more of the semantic/OCL constraints and some form of the behavioral portion of the UML Semantics Guide.

{ Add more references e.g. Lano here }

## References


[UMLSEM] Booch, G., Rumbaugh, J., Jacobsen, I.: *UML Semantics, Version 1.1, Rational Software Corp.*, Sept. 1, 1997.

[W+98] R. Waldinger et al.: *Specware Language Manual: Specware 2.0.2*, 1998.

[BC95] Bourdeau, R., Cheng, B.: *A Formal Semantics for Object Model Diagrams*, IEEE Transactions on Software Engineering, 21(10):799-821, Oct. 1995.

[DeL96] S. DeLoach: *Formal Transformations from Graphically-Based Object-Oriented Representations to Theory-Based Specifications*, PhD Thesis, Air Force Institute of Technology, June 1996, PhD Dissertation.

[MF97] M. Fowler: *UML Distilled*, Addison Wesley, pg. 80, 1997.

[Spe98] *Specware Language Manual, Version 2.0.3*, March 1998.