Category Theoretic Approaches of Representing Precise UML Semantics

Jeffrey Smith[1], Scott DeLoach[2], Mieczyslaw Kokar[3] and Ken Baclawski[3]

[1] Mercury Computer Systems Inc.
[2] Northeastern University
[3] Air Force Institute of Technology

1 Introduction

There have been a number of formal approaches for specifying UML semantics [2–4, 8–11]. Unlike these approaches, the purpose of this position paper is to present a case for the community defining precise UML semantics to consider category theoretic approaches for representing UML semantics based on the authors actual experiences. Category theory has been gaining popularity for formal approaches in software engineering. While many formal specification techniques provide the ability to describe the structure and behavior of specification objects, category theory explicitly captures relationships between specification objects. The motivation for considering a category theory based approach is:

- At least a portion of the pUML group is searching for diagrammatic techniques for formal semantic representations based on last year's workshop held concurrent with OOPSLA'99.
- To have tool support for developing domain theories (ontological engineering), code from specifications, specifications from code and specification refinement.
- The graphical and language support for specification composition is much more powerful than Z-like *includes* or with simple algebraic specification techniques.
- The general theory of diagrams enables nodes and arcs to be related in a way that preserves the structure and content of the source diagram in the target diagram making it possible for diagrams of specifications to be parameterized and instantiated where the result is a diagram, not a single specification [12].
- Category theory is intensional versus extensional - implicit versus explicit [5], i.e. to check an instance of a UML diagram against a category theory based form of semantics one would show that the class of models of any instance is within the class of models of the UML metamodel, rather than show an explicit association (of classes of models with algebraic specifications in both the source and target specifications).
- Category theory has been described as a potential formal foundation for the emerging UML RT standard [7].

DeLoach [13] motivated the research described in this position paper by specifying OMT semantics using category theory arrows to map from the internal structure between category objects (objects in the category of interest), so specification morphisms can map the axioms in a specification to theorems of the derived specification. In this research, DeLoach had developed an object oriented extension of Kestrel Institute's Slang [17] - an algebraic

specification language where specifications are theory presentations using higher order logic extended with category operations e.g. products, coproducts, quotients, subsorts and diagrams. This SLANG extension, called O-SLANG, was developed to incorporate the category theory operations necessary to define relationships between object classes.

We've constructed two methods of implementing correctness proving transformations from UML to a SLANG formal specification. One approach was to construct a SLANG form of a Core metamodel portion of the UML Semantics Guide [1] (by coincidence, this Core metamodel is very much like the one defined by the pUML group in [4]) and to ensure that a transformation from a UML application to SLANG is consistent with the SLANG form of UML Semantics [15, 16]. Another approach was to define a theory-based object model, with an associated object-based formal specification language (viz. O-SLANG) and to translate UML applications to O-SLANG. In a related effort, we've performed the language-to-language translation of O-SLANG to SLANG. The first approach is distinguished from the second in that,

- The focus was to definitize UML through formalizing the UML metamodel rather than creating a theory-based object model that could be used for other object oriented specification paradigms.
- The goal was on the meta-modeling and validation process and tool support, generalizable to formalization of many semi-formal modeling languages.
- An object and instance are both meta-objects, specified in the UML Semantics Guide, and are dependent on a meta-object hierarchy that includes specifications of many other meta-objects, e.g. Classifier, Association, Attribute, etc., all of which have only semi-formal semantic specification.

In either method, the translation of UML diagrams to formal semantics was automated. In this paper, we give a brief overview of each technique and pointers to the complete research descriptions to further a workshop discussion on the use of category theory as a formalism that could be used to represent UML semantics. The next step would be to compare our two implementation approaches with other similar category theoretic efforts and work with the community defining precise UML semantics, contributing portions of an agreed upon rigorous UML semantic description of consensus.

2 Method 1 - Category Theory Based Form of UML Semantic Representation and Validation Overview

The purpose of this research was to bridge formal and CASE-based development methods by deriving a methodical process for checking the correctness of the automatic translation of a UML diagram to a formal language with respect to the UML metamodel. This process consisted of formalizing a subset of the UML metamodel and showing that the class of models generated by the translator was within the class of models of the UML metamodel.

The formalization of both UML, and UML application specifications, must be performed in an extensible and automatable fashion that supports specification composition and CASE

tool interoperability. The focus of this research was in the meta-modeling and validation process and tool support generalizable to the formalization of many semi-formal modeling languages. This UML formalization process approach is portrayed in Figure 1.

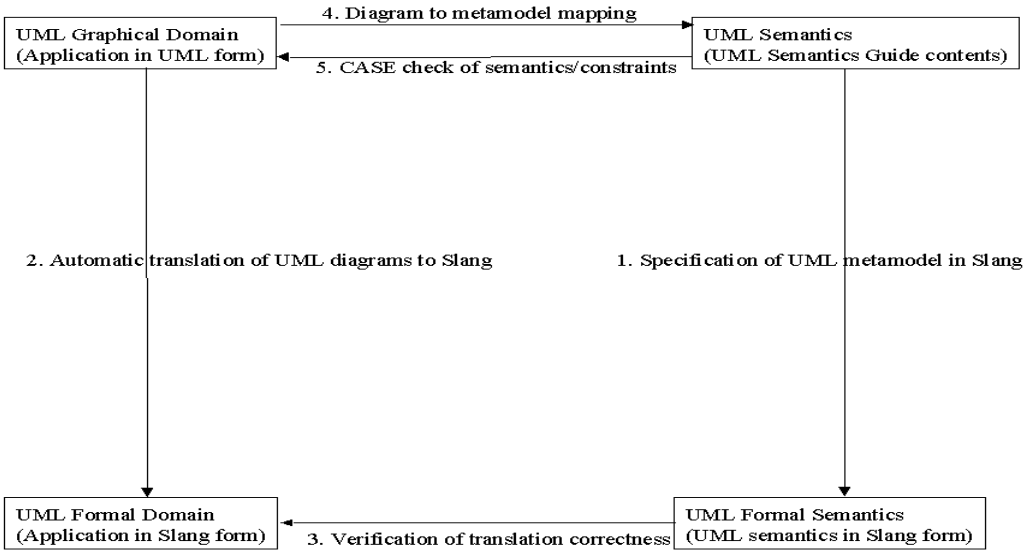

Fig. 1. UML Semantic Formalization Process

Mapping 1 describes the formalization of the UML metamodel. Formalization rules describe how to translate UML Semantics Guide (meta) objects into SLANG, with a rationale for each formalization rule and a description of formalization rule alternatives.

Mapping 2 shows the software and tool support needed to automatically translate UML applications (described as the UML Graphical Domain) to SLANG, according to the formalization rules described in the last paragraph. The SLANG version of the UML Graphical Domain is called the UML Formal Domain.

Mapping 3 represents the verification of the correctness of the translation. Here, instances of the SLANG form of the UML Graphical Domain (i.e. the UML Formal Domain) are checked for compatibility with the SLANG representation of abstract UML theory, viz. the UML Formal Semantics in Figure 1. Instances are checked to ensure that the SLANG form of the UML Graphical Domain translation preserves the UML semantics captured in mapping 1. To explain this step, both the specification of the UML metamodel, and of any UML diagram, are viewed as a presentation of a theory (in SLANG). The goal is to ensure that the class of models of the theory, obtained as a translation of any UML diagram, is a subclass of models of the theory of the UML metamodel. In order to show this, it must be shown, for each such translation, there exists a morphism from the UML metamodel theory to the theory representing a given UML diagram.

Mapping 4 is a mapping of all possible UML diagrams that can be produced within a UML CASE tool into the UML metamodel. In other words, mapping 4 represents an *interpretation* of a UML diagram in the UML metamodel. Mappings 1 and 4 collectively constitute the formalization of UML diagrams. Mapping 5 shows the check of UML semantics and constraints that is assumed to be performed within a CASE tool.

3 Method 2 - Theory-based Specification Overview

The goal of the theory-based specification method was to transform graphically based object-oriented diagrams into working code. Our approach was to define a theory-based object model that captured all the important characteristics of object-orientation without regard to the actual UML Semantics. While not actually part of the translation process, the theory-based object model was central to the research as it defined how we could represent object-oriented concepts in an algebraic specification language. The translation then became a matter of mapping UML concepts to the generically defined object model. In addition, because it is not strictly tied to the UML semantics, the theory-based object model can be used for other object-oriented techniques and methods (e.g. OMT).

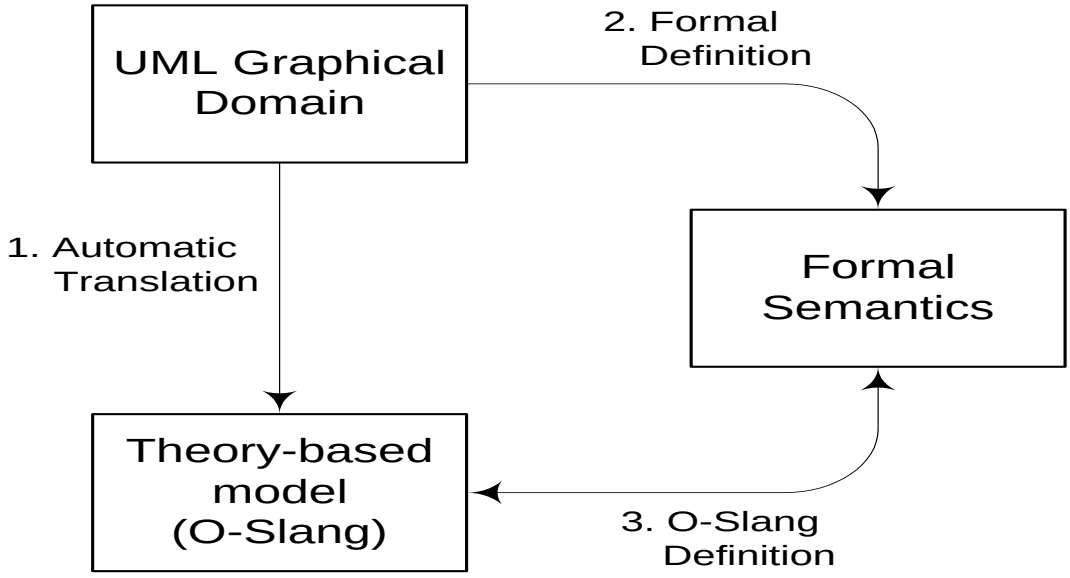

Fig. 2. Theory-based Translation

However, to show that the translation from UML to the theory-based model was correct, we had to show that it preserved the semantics of the UML diagrams. Since no formal semantics had been established, we defined our own. The basic approach is shown in Figure 2.

Translation 1 is the automated translation from UML to O-Slang as described in [14]. This translation maps each construct in a UML diagram to an equivalent construct in O-Slang. This transformation is defined formally and implemented in software.

Translation 2 is the formal definition of UML diagrams. This can be any formal definition that suits the purpose. For instance, for Statechart Diagrams we used an extension to the standard automata theory definitions.

Translation 3 is a bijective mapping from O-SLANG to the same formal definitions used for the UML diagrams. Once these mapping were complete, we showed that the transformation via translation 1 was equivalent to the mapping of translation 2 composed with translation 3. Translation 1 was then implemented and shown to be a practical approach to generating formal specifications from graphical object oriented diagrams.

4 Conclusion

Despite the differences mentioned in the introduction section, the two methods described in this paper have a lot in common. In Method 2, the underlying system model was based on a combination of algebraic and category theory based specification methods - a mathematical base that was used to help implement and enforce software/specification in Method 1. Both approaches also used Kestrel's Specware to demonstrate a thread of each complete approach for proof-of-concept. The research described with both methods primarily differs in the foundation for formalization. The formalization in Method 2 was based on a theory-based object model DeLoach formulated. The research in Method 1 used a formalized form of the UML Semantics Guide as an object model.

Other groups are also looking at category theory to describe UML (or other object oriented systems) [6, 7]. The authors are aware of other activities that may become mature enough to also discuss at ECOOP. While the pUML group is exploring rigorous methods of representing UML semantics with significant tool support, it would be worth exploiting the lessons learned from the cited research in this paper in upcoming UML formalization efforts, e.g. the one described with UML RT.

References

1. G. Booch, J. Rumbaugh and I. Jacobsen, *UML Semantics, Version 1.1, Rational Software Corp.*, 1997.
2. R. Breu and U. Hinkel et al: *Towards a Formalism of the Unified Modeling Language*, ECOOP '97, pg. 344-366, 1997.
3. A. Evans, R. France, K. Lano and B. Rumpe: *The UML as a Formal Modeling Notation*, pUML Working Group, 1998.
4. A. Evans and S. Kent: *Core Meta-Modeling Semantics of UML: The pUML Approach*, UML99 - The Unified Modeling Language: Beyond the Standard, Second International Conference Proceedings, Springer, ISBN 3-540-66712-1, October 1999.
5. J. Fiadeiro and T. Maibaum: *Category Theory for the Object Technologist*, OOPSLA-94, ftp://ftp.fc.ul.pt/pub/papers/modeling/94-FM-OOPSLA-TutCateg.ps.gz.
6. J. Fiadeiro and T. Maibaum: *Describing, Structuring and Implementing Objects*, Rex90 Workshop on the Foundations of Object Oriented Languages (eds. J. de Bakker, W-P de Roever, G. Rozenberg), LNCS 489, pp. 274-310, Springer-Verlag, 1991.
7. R. Grosu, M. Broy, B. Selic and G. Stefănescu: *What is Behind UML-RT*, ISBN 0-7923-8629-9, Kluwer Publishing, 10/99.
8. S. Kim and D. Carrington: *Formalizing the UML Class Diagram Using Object-Z*, UML'99 - The Unified Modeling Language: Beyond the Standard, Department of Computer Science and Electrical Engineering, The University of Queensland, Brisbane, Australia, Lecture Notes in Computer Science, Springer, ISBN 3-540-66712-1, 10/99.

9. K. Lano and J. Bicarregui: *Formalising the UML in Structured Temporal Theories*, Seventh OOPSLA Workshop on Behavioral Semantics of Object Oriented Business and Systems Specifications, TUM-I9813, Aug. 1998.
10. G. Övergaard: *The Semantics of the Unified Modeling Language - Tutorial at OOPSLA '96*, San Jose, Oct. 1996.
11. J. Robbins, N. Medvidovic, D. Redmiles and D. Rosenblum: *Integrating Architecture Description Languages with a Standard Design Method*, White paper based on work sponsored by NSF grants CCR-9924846 and CCR-9701973 and DARPA, RL and USAF, University of CA, Irvine.
12. T. Schorsch: *Formal Representation and Application of Software Design Information*, PhD thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, September 1999.
13. S. A. DeLoach, *Formal Transformations from Graphically-Based Object-Oriented Representations to Theory-Based Specifications*, PhD Thesis, Air Force Institute of Technology, June 1996, PhD Dissertation.
14. S. DeLoach, T. Hartrum and J. Smith, *A Theory-Based Representation for Object-Oriented Domain Models*, IEEE Transactions on Software Engineering, 1999. (to appear).
15. J. Smith, M. Kokar and K. Baclawski: *Formal Verification of UML Diagrams: A First Step Towards Code Generation*, OOPSLA'99, November 1999.
16. J.Smith, *UML Formalization and Transformation*, Ph.D. Thesis, Northeastern University, College of Engineering, 1999.
17. R. Waldinger et al, *SPECWARE Language Manual: Specware 2.0.3*, Kestrel Institute, 1998.