Towards a Formal Framework for Multimodeling in Software Engineering

Rick Salay

Supervisors: John Mylopoulos, Steve Easterbrook
University of Toronto
{rsalay, jm, sme}@cs.toronto.edu

Technical Problem

In Software Engineering, we typically model systems using multiple "partial" models of different types. There are several reasons for this. Firstly, different views of the software are best captured using a modeling language most appropriate for that view. Secondly, the complexity of a model for a piece of software requires that it be decomposed into smaller parts and descriptions at different levels of abstraction. Thirdly, the widely accepted principle of "separation of concerns" requires that different models be created to address different purposes. Finally, different models can express the viewpoints of different stakeholders. Furthermore, the use of multiple models is typically supported by the development process - most contemporary modeling paradigms (e.g. UML) and development processes mandate the use of multiple views and require some form of iteration in which a series of progressively more detailed models are created.

Although the use of multiple models is necessary for all of the reasons listed above, it raises the problem of how to effectively work with such a set of interrelated models – i.e. how to comprehend, create, check, extend, change or otherwise manipulate them in meaningful ways to achieve certain modeling objectives. Thus, tools are clearly required to assist with this and such tools should be based on a suitable formalism. My research objective is to define such a formalism and illustrate its use in example tools.

Research Claim and Approach

A key observation that motivates my work is that the relations between models are seldom just generic "mappings" but instead usually realize an incremental modeling step of some kind. Thus, we have steps like translations, projections, refactorings, refinements, decompositions, merges, the taking of sub-models or aspects, etc. In each case, the relation contains the details of how the elements of the component

models in the step are related[1] . These details constitute the syntactic and semantic aspects of a relation while the modeling step enacted by it is its "pragmatic" aspect.

In order to provide tool support for modeling with many models, a formalism is required that treats model relations and sets of interrelated models, including their pragmatic aspects, as first class entities that can be typed, characterized using metamodels, reasoned about and manipulated using operators. To achieve this I propose an approach with two key facets. Firstly, a set of interrelated models can be viewed as a kind of hierarchical model – a *multimodel*. Secondly, relations types can be classified using meta-types corresponding to the typical modeling steps that arise in software engineering. Together, these provide a unified framework in which to express modeling scenarios within software engineering. We now elaborate these facets further.

A metamodel can be used to define a relation type between model types by showing what element types are found in an instance of the relation type and how these are used to relate the elements of the related models. If we then define a hierarchical metamodel as consisting of metamodels for a set of model types and relation types between them (and possibly additional constraints), then an instance of this is a set of models and relations between them that conform to the metamodels. We call such a set, a *multimodel*, and this kind of metamodel defines a multimodel type. More generally, we can define an order hierarchy of models – elements such as "class", "component" and "state" are considered 0^{th} order models, models such as class diagrams and statecharts that consist of these elements and their relations are 1^{st} order models, models consisting of 1^{st} order models and their relations are 2^{nd} order models, etc. Relations have a more complex "order arity". For example, the relation type that relates a class to its statechart is a (0, 1)-order relation since it relates 0^{th}-order and 1^{st}-order model while the sub-model relation that relates a class diagram to another that contains it is a (1,1)-order relation. An example of a (2,1)-order relation is one where a set of class diagrams with relations between them is related to another class diagram that represents their "merge." Thus, a multimodel can contain models and relations of different orders and provides the necessary richness to express the structure of a complex modeling scenario. In addition, the underlying constraint language (we use order sorted first order logic) allows for various types of reasoning including the checking of the static semantics, consistency checks between constituent models by checking the static semantics of relations, inference of relations from other relations, etc.

Since a multimodel is hierarchical, it naturally lends itself to different abstractions based on aggregation. We refer to an abstraction such as this as a *macromodel*. Thus, a macromodel is a graphical model whose elements denote models and whose edges denote model relations. Macromodels are useful both because they are convenient views of the structure of a multimodel and also because they can be used to specify a multimodel. The latter is achieved by considering some model/relation elements to be placeholders that denote future models/relations to be created and these are constrained by the indicated typing and their relations to other existing models. For example, a placeholder for a sequence diagram may indicate that it must reference

[1] Note that many of these steps are not "transformations" because the result model cannot be generated from the source model, but they are generally directed relations.

classes in a particular existing class diagram and must refine a particular existing sequence diagram.

In addition to supporting model relation types as first class entities, we use syntactic, semantic and pragmatic aspects of relation types to classify them using meta-types. For example, the generic signature *Transformation*(*M, M1*) specifies a binary relation meta-type with instances being relation types such as the UML to Java transformation UML2Java(UML, Java). The key syntactic constraint of a transformation is that the *M1*-model must be uniquely determined once the *M*-model is given - thus, it acts like a function and we can also write it as *Transformation*(*M*)→*M1*. Some common subtypes of *Transformation*(*M*)→*M1* include *Projection*(*M*)→*M1* and *Translation*(*M*)→*M1* which have additional syntactic and semantic constraints. Other relation meta-types include *Refinement*(*M*, *M1*), *Refactoring*(*M*, *M*), *Submodel*(*M*, *M*), *Aspect*(*M*, *M1*), *Homomorphism*(*M*, *M*), etc. These meta-types can be further qualified by the orders of the models they deal with. For example, *Refinement*(M^0, $M1^1$) classifies relations that decompose a 0^{th} order model type (i.e. an element type) into a set of related elements represented as a 1^{st} order model type while *Merge*(M^2) → $M1^1$ classifies transformations that merge sets of interrelated 1^{st} order models into another 1^{st} order model.

The key contribution of this framework is to provide a uniform approach for expressing different modeling scenarios that arise in software engineering. In particular, the fact that it is metamodel driven makes it applicable in a wide variety of situations including model driven engineering scenarios based on multi-view modeling languages such as UML or for related sets of domain specific modeling languages.

Related Work

Existing work on dealing with multiple models has been done in a number of different areas. Metamodeling is a key component of any such formalism. The foundational work of Telos [12] on metamodeling within software engineering defined a very general approach to modeling at multiple meta-levels but did not address the definition of multiple model types. More recently, configurable modeling environments, sometimes called meta-CASE tools, allow model types to be defined and corresponding modeling tools to be automatically generated for creating and processing the models. These include the Eclipse Graphical Modeling Framework (GMF) [7], Generic Modeling Environment (GME) [13], Domain Modeling Environment (DoME) [5] and MetaEdit+ [10]. In each case, there is a metametamodeling language that is used for defining metamodels. All approaches allow the use of metamodels for defining model types containing simple elements and relations but not necessarily models containing models. This is the case with MOF and Ecore. In contrast, the MetaEdit+, DoME and GME tools do support models containing models – possibly because they emerged in the context of work in Method Engineering where the focus is on defining a single metamodel that encompasses an entire development method. However, even with these, model relations are not treated

as first class entities and cannot be defined in the rich ways that are required for general multimodeling.

The emerging field of Model Management [2] has close ties to my work. This has developed in response to the problems of dealing with multiple models in meta-data management such as the schema integration problem. A key part of the approach here is to express the relation between two models by defining a "mapping" between them and then treating models and mappings as basic units that can be manipulated at the macroscopic level by using a set of generic model management operators. This basic idea has been elaborated in various ways [6, 3, 4]. On the one hand, my focus is different than this work in that I am interested in supporting the modeling process whereas the motivation behind model management is primarily model integration. As a result, I have a richer taxonomy of model relation types and view model management operators as particular subclasses of the transformation meta-type. On the other hand, this work has strong mathematical foundations that may be of value in my framework. This is something I am investigating.

The concept of a macromodel is similar to that of a "megamodel" as first defined by Favre[8] and also later as part of the Atlas Model Management Architecture (AMMA) [1]. In both cases the elements of a megamodel denote models and the edges denote the relations between them. My approach differs from these in that a macromodel is type of model related to a multimodel in a formal way via abstraction, whereas a megamodel is closer to a form of documentation for a resource repository used in modeling.

Progress and Methods

I have developed an initial candidate metamodeling formalism for multimodels based on sorted first order logic with transitive closure and use some facets of Institution theory [9]. An initial taxonomy of relation meta-types has also been proposed based on a survey of the software engineering literature. The intention is to refine this further in the context of actual usage scenarios.

In order to do some preliminary evaluation and experimentation with the framework, a hypothetical metamodel was defined for UML multimodels (called "UMLLite") and three multimodeling use cases were defined and tested on a set of UML diagrams sourced from a publicly available example project [11]. The first use case was to extend the set of diagrams to a multimodel based on the UMLLite metamodel by adding the model relations between the diagrams as specified in the metamodel. The macromodel was then produced to show the structure of this multimodel. The goal here was to informally determine whether a macromodel could be useful for making the set of models more comprehensible and this indeed seemed to be the case. A more rigorous evaluation is required to determine whether this is generally true. The second use case involved showing that this macromodel could be used to specify extensions to a multimodel. This experiment revealed that even a simple specification task sometimes needs to use relations between relations. Finally, the third use case was to show how to develop an operator (i.e. transformation) for constructing new sequence diagram refinements from existing ones. The conclusion

of this experiment was that, although such an operator could be "coded" using axioms in the metamodeling formalism, it is cumbersome and it suggests that higher order extensions to the logic would be desirable. I am investigating this.

In order to actualize and evaluate the framework in a more in-depth way, I am developing an Eclipse-based tool to implement the framework. The intention is to use this as a basis for doing more detailed case studies that will help evaluate the usefulness of the framework.

References

[1] "ATLAS MegaModel Management" website: http://www.eclipse.org/gmt/am3/

[2] Bernstein, P. A. "Applying Model Management to Classical Meta Data Problems," In *Proc. CIDR*, 2003

[3] Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., and Sabetzadeh, M. 2006. "A manifesto for model merging." In *Proceedings of the 2006 international Workshop on Global integrated Model Management* (Shanghai, China, May 22 - 22, 2006). GaMMa '06. ACM Press, New York, NY, 5-12.

[4] Boronat, A., Carsí, J. A., Ramos, I. "An Algebraic Baseline for Automatic Transformations in MDA," *Electr. Notes Theor. Comput. Sci*. 127(3): 31-47 (2005)

[5] DoME website. http://www.htc.honeywell.com/dome/support.htm#documentation

[6] Diskin, Z. "Mathematics of Generic Specifications for Model Management I," In *Encyclopedia of Database Technologies and Applications* 2005: 351-358

[7] Eclipse Graphical Modeling Framework website. http://www.eclipse.org/gmf/

[8] Favre, J. M. "Modelling and Etymology." Transformation Techniques in Software Engineering 2005

[9] Goguen, J.A. and Burstall, R.M. "Institutions: Abstract Model Theory for Specification and Programming." *J. ACM* 39(1): 95-146, 1992.

[10] MetaEdit+ website. www.metacase.com

[11] *Methods for Testing and Specification (MTS); Methodological approach to the use of object-orientation in the standards making process*. ETSI EG 201 872 V1.2.1 (2001-08): http://portal.etsi.org/mbs/Referenced%20Documents/eg_201_872.pdf

[12] Mylopoulos, J.; Borgida, A.; Jarke, M.; Koubarakis, M. "Telos: Representing Knowledge About Information Systems", TOIS 8(4), pp. 325-362.

[13] "The Generic Modeling Environment." http://www.isis.vanderbilt.edu/publications/archive/Ledeczi_A_5_17_2001_The_Generi.pdf