Managing Models through Macromodeling

Rick Salay John Mylopoulos Steve Easterbrook
Department of Computer Science
University of Toronto
Toronto, ON M5S 3G4, Canada
{rs, jm, sme}@cs.toronto.edu

Abstract— **Software development involves the creation and use of many related models yet there are few tools that address the issue of how to work with and manage such collections of models, or "multimodels." We propose a formal multimodeling framework that allows specialized model relationship types to be defined on existing types of models and provides a new type of model with a formal semantics called a macromodel. Macromodels can be used to enhance multimodel development, comprehension, consistency management and evolution. A preliminary evaluation of the framework is done using a detailed example from the telecommunications domain.**

Model relationships; Multi-view modeling; Metamodeling; Model consistency

I. MOTIVATION

Existing approaches to working with multiple interrelated models (or *multimodeling*) tend to focus on how the contents of specific model instances are related (e.g. [7, 9]), rather than on the relationship types themselves. Those approaches that do provide abstractions for expressing relationship types typically concentrate on a limited set of relationships required to support model transformation (e.g. [2, 4]) and/or traceability (e.g. [1]). In contrast, we argue that a rich, extensible set of relationship types for relating models is needed, to capture the intended meaning of the models and the various ways that models in a multimodel constrain one another.

UML suffers from a similar limitation. The UML metamodel defines a number of diagram types as projections of a single UML model. However, developers rarely manipulate the underlying UML model directly - they work with the diagrams, and the intended purpose of each new diagram is only partially captured in the UML metamodel. For example, a developer might create an object diagram showing instantiations of a subset of the classes, chosen to be relevant to a particular use case scenario. The relationships of these objects to the classes in the model is captured in the metamodel, but the relationship between this particular object diagram and the set of behaviour models (e.g. sequence diagrams) that capture the scenario is left implicit.

The key point is that each model in a multimodel is created for a specific purpose, for example to capture important concepts, or to elaborate and constrain other models. Each model therefore has an *intended purpose* and a set of *intended relationships* with other models. Such relationships might include submodels, refinements, aspects, refactorings, transformations, instantiations, and so on.

In this paper we describe a framework for systematically extending a modeling paradigm to formally define model relationship types between model types and we introduce a new type of model, called a *macromodel*, for managing multimodels at a high level of abstraction expressed only in terms of models and their intended relationships.

The framework provides a number of benefits:

1. Understandability. Making the underlying relational structure of a set of models explicit helps comprehension.
2. Specifying constraints on models, even for models yet to be created. When constituent models change, they are still expected to maintain the relationships expressed in the macromodel, because the macromodel captures the intentions of the modelers on how the set of models should be structured.
3. Consistency checking. The macromodel can be used to assess an evolving set of models for conformance with modeler intentions even when mappings between models are incomplete.
4. Multimodel evolution. As the macromodel itself evolves, it can be taken as a specification for change in the multimodel – either involving the removal of some models, changes in model relationships, or additions of new models.

The structure of this paper is as follows. In Section II we describe the formal approach to defining relationship types, macromodels and multimodels. Section III provides a preliminary evaluation of the framework. Section IV outlines related work and Section V discusses conclusions and tool development.

II. FRAMEWORK DESCRIPTION

The application of the framework to a development paradigm involves the following steps:

1. Define the relationship types that are required for relating model types.

2. Using the relationship and model types, define the metamodel for multimodels in this paradigm. This also defines the metamodel for the macromodels that will be used.

3. For a given development project within the paradigm, define an initial macromodel (according to the metamodel) expressing the models and their relationships. As the project continues, both the macromodel and the constituent models continue to evolve. During the project, the macromodel is used as a tool for supporting the comprehension of the project multimodel, specifying extensions and checking degree of conformance.

We describe the formal basis for each of these steps below.

A. *Model relationship types*

The problem of how to express relationships between models has been studied in a number of different contexts [9, 10, 3]. At the syntactic level, this relationship can be expressed by embedding the models within a larger *relator* model that contains the mapping showing how the elements of the models are related. For example, Fig. 1 shows the (simplified) metamodels defining the *objectsOf* relationship between a sequence diagram and an object diagram. The *objectsOf* relationship holds when the object diagram contains the links over which the messages of the sequence diagram are sent. The well-formedness rules of the *objectsOf* metamodel (i.e. the axioms) define the consistency constraints that must hold between the related models. For example, it includes the constraint that when it maps a message to a link, then the start and end objects of the message must be the start and end objects of the link, respectively.

We call the mappings p_{OD} and $p_{SD,}$ *metamodel morphisms*. These are used to relate metamodels in a way that maps the elements and preserves well-formedness rules. We exploit the similarity of metamodels to logical theories and employ elements of Institution theory [5] in order to formally define these mappings. The key benefit of using metamodel morphisms for relating metamodels is that the approach can be formulated in a way that is independent of the metamodeling language and can also be used to relate metamodels expressed in different metamodeling languages.

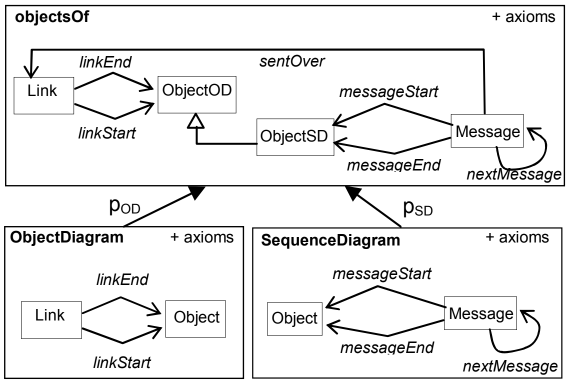

Figure 1. Metamodels mapped to define *objectsOf*

In addition to "custom defined" relationship types such as *objectsOf* there are a variety of useful generic parameterized relationship types that can be automatically constructed from the metamodels for the associated model types. We discuss two of these briefly as they are used in Section III.

Given any model type T, we can define the relationship type *eq*[T] where *eq*[T](M_1, M_2) holds iff there is an isomorphism between M_1 and M_2, where we interpret corresponding elements as being semantically equal – i.e. they denote the same semantic entities. The second parameterized type we define is *merge*[T]. Given a multimodel K consisting of T-models, *merge*[T](K, M) holds iff M is the smallest T-model that contains all of the models in K as submodels. Note that a unique merge M may not always exist.

B. *Macromodels and multimodels*

With the notion of a relationship type in hand, we can now define macromodels and multimodels. A macromodel is a hierarchical model whose elements are used to denote models and model relationships and can express integrity constraints on these. A multimodel is defined as a macromodel with a collection of models/relationships that conform to their types and in addition, conform to the constraints in their macromodel.

Fig. 2 depicts a simple example of this. The upper part is the multimodel metamodel *SimpleMulti* consists of a metamodel for macromodels (left side) and a set of metamodels for the model and relationship types (right side). The axioms of the macromodel limit the possible well-formed collections of models and relationships that can occur in a multimodel of this type.

The lower part of Fig. 2 depicts a particular multimodel *M*:*SimpleMulti* that conforms to the metamodel. This consists of a macromodel and a set of models and relationships that conform to it. The asterisk preceding *f*:*objectsOf*, *M2*:*ObjectDiagram* and *f1*:*refines* indicate that they are "unrealized" models and relationships. This implies that there is no corresponding instance for these in the multimodel and

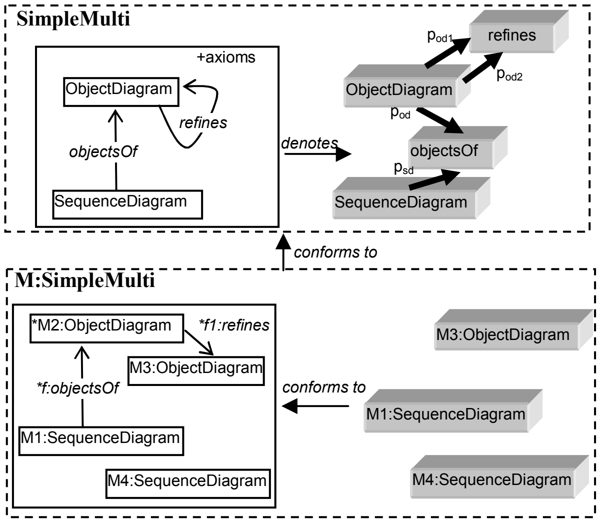

Figure 2. A multimodel metamodel and an instance of it

they are just placeholders. A model or relationship may be unrealized for two main reasons: they may represent models/relationships that are required to be created but do not yet exist or they may be models/ relationships that are derived from others and hence need not be materialized. The macromodel in Fig. 2 expresses the constraint that “M4 is a sequence diagram and the object diagram corresponding to sequence diagram M1 is a refinement of object diagram M3.” Translated into first order logic we have the following set of sentences:

SequenceDiagram(M4),
SequenceDiagram(M1),
ObjectDiagram(M3),
$\exists f, f1, m2.$ objectsOf(f) $\wedge$ $p_{sd}(f)$ = M1 $\wedge$ $p_{od}(f) = m2$
$\wedge$ refines($f1$) $\wedge$ $p_{od1}(f1) = m2$ $\wedge$ $p_{od2}(f1)$ = M3
$\wedge$ ObjectDiagram($m2$)

Here, projection functions p_i correspond to metamodel morphisms. Also, each connected set of unrealized elements in the macromodel is translated to an existential sentence with the unrealized models and relationships as existentially quantified variables. The full definition of macromodel semantics as a translation into first order logic is omitted here due to lack of space – please see [12] for details.

Expressing macromodel semantics as a set of sentences in first order logic provides a way to check multimodel conformance using standard model checking tools. Furthermore, different degrees of conformance between a multimodel and a macromodel can be assessed by determining the subset of sentences that the multimodel satisfies. This addresses the reality of incremental model development in that it allows for the case in which some or all the models and relationships may be incomplete within a version of the multimodel.

III. Preliminary Evaluation of the Framework

We applied the approach to an example design project taken from a standards document for the European Telecommunications Standards Institute (ETSI) [8]. The example consists of 42 diagrams over three UML models and details the development of the Private User Mobility dynamic Registration service (PUMR) – a simple standard for integrating telecommunications networks in order to support mobile communications.

Fig. 3 shows the multimodel metamodel *UMLMulti* that we constructed and Fig. 4 shows part of the macromodel containing relationships both between the diagrams within each UML model and also between the diagrams across the models. Due to lack of space we do not show the definitions of the relationship types of *UMLMulti* - see [12] for further details.

We found two interesting cases where we needed to express complex relationships using unrealized models. The relationship between sequence diagram 42 and class diagram 41 is expressed using the unrealized object diagram *42a and this is also used to show that object diagram 43 is a refinement of the objects in diagram 42. Another example is the one between the three sequence diagrams 53, 54, 55 and the object diagram 52. The macromodel shows that 52 is the smallest superset (i.e. the merge) of the object diagrams corresponding to each of these sequence diagrams.

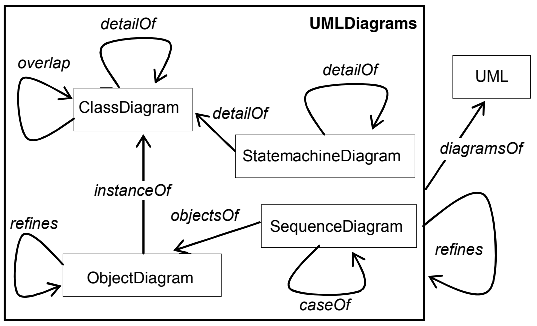

Figure 3. UMLMulti

Even without understanding the details of the domain, it should be clear how the expression of the relationships between diagrams help to explicate the underlying structure in this multimodel. We conclude from this example that the approach is adequate for expressing the kinds of relationships needed for the PUMR domain and that creating this macromodel aids the comprehension of the multimodel. Since this example involves an existing completed multimodel, we are not able to assess the hypothesis that the macromodel can be used throughout the development lifecycle to assess conformance and guide development. In order to do this, we are planning to do an in-depth case study over an entire project lifecycle.

IV. Related Work

Existing work on dealing with multiple models has been done in a number of different areas. The ViewPoints framework [9] was an influential early approach to heterogeneous multimodeling. Our approach differs from this work in being more formal and declarative rather than procedural.

More recently, configurable modeling environments have emerged such as the Generic Modeling Environment (GME) [7]. None of these approaches provide general support for expressing model relationships or their types; hence, they have limited support for defining and expressing multimodels. Furthermore, the focus of these approaches is on the detail level (i.e. the content of particular models) rather than at the macroscopic level.

Process modeling approaches like the Software Process Engineering Metamodel (SPEM) [13] bear some similarity to our notion of a multimodel metamodel; however, our main focus is in the use of a macromodel at the instance level to allow fine-grained control over the ways in which particular models are related. Process modeling approaches have no mechanism corresponding to this.

The emerging field of Model Management [3] has close ties to our work but our focus is different in that we are interested in supporting the modeling process whereas the motivation behind model management is primarily model integration.

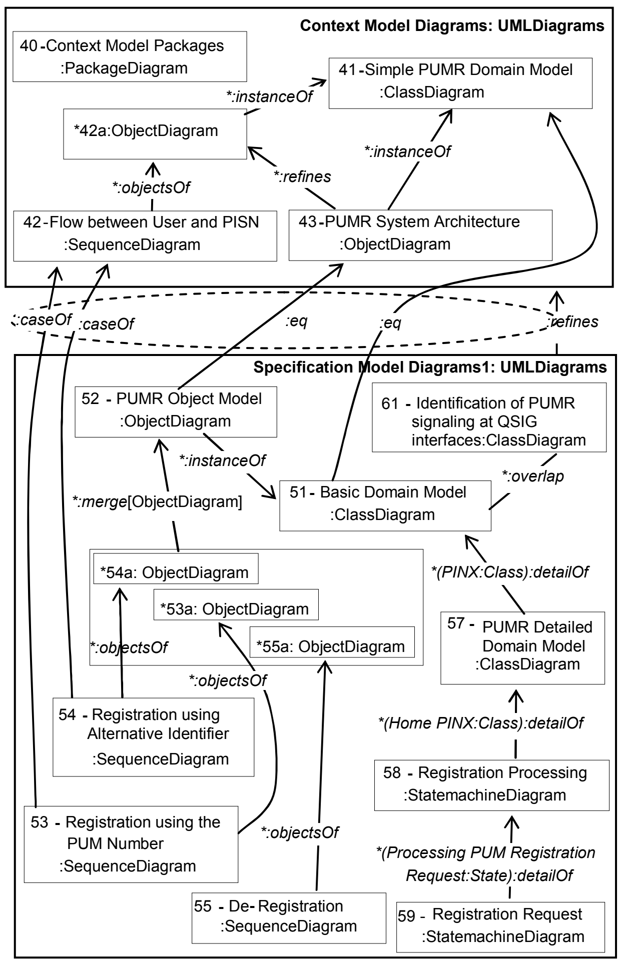
Context Model Diagrams: UMLDiagrams
40-Context Model Packages :PackageDiagram
41-Simple PUMR Domain Model :ClassDiagram
*:instanceOf
*42a:ObjectDiagram
*:instanceOf
*:refines
*:objectsOf
42-Flow between User and PISN :SequenceDiagram
43-PUMR System Architecture :ObjectDiagram
:caseOf
:caseOf
:eq
:eq
:refines
Specification Model Diagrams1: UMLDiagrams
52 - PUMR Object Model :ObjectDiagram
61 - Identification of PUMR signaling at QSIG interfaces:ClassDiagram
*:instanceOf
*:overlap
*:merge[ObjectDiagram]
51 - Basic Domain Model :ClassDiagram
*54a: ObjectDiagram
*53a: ObjectDiagram
*55a: ObjectDiagram
*(PINX:Class):detailOf
*:objectsOf
*:objectsOf
57 - PUMR Detailed Domain Model :ClassDiagram
54 - Registration using Alternative Identifier :SequenceDiagram
*(Home PINX:Class):detailOf
58 - Registration Processing :StatemachineDiagram
53 - Registration using the PUM Number :SequenceDiagram
*:objectsOf
*(Processing PUM Registration Request:State):detailOf
55 - De-Registration :SequenceDiagram
59 - Registration Request :StatemachineDiagram

Figure 4. Telecommunications example macromodel

The term "megamodel" as representing models and their relationships at the macroscopic level emerged first in the work of Favré [4] and also later as part of the Atlas Model Management Architecture (AMMA) [2]. Macromodels bear similarity to these two kinds of megamodels, but the intent and use is quite different – to express the modeler's intentions in order to specify multimodel structure in particular development processes.

Finally, the work on model traceability also deals with defining relationships between models and their elements [1]; however, this work does not have a clear approach to defining the semantics of these relationships. Thus, our framework can provide a way to advance the work in this area.

V. Conclusions and Tool Development

In this paper we have described a formal framework for extending a modeling paradigm with a rich set of model relationship types and for using macromodels to manage multimodels at a high level of abstraction. A general, metamodel-language independent approach for defining relationship types was presented that defines a relationship type as a metamodel with special mappings called metamodel morphisms connecting the metamodels of the model types it relates.

Some aspects of our framework have been incorporated into the Eclipse-based Model Management Tool Framework (MMTF) [11] and we consider this a first phase of tool development. As part of current work, we are exploring the use of model finders such as Alloy [6] and the use of a macromodel as a "constraint problem" in order to do model synthesis. A key benefit of this is to allow model consistency checking to be performed with incomplete mappings by checking whether there exist extensions to the mapping that satisfy the macromodel constraints.

References

[1] N. Aizenbud-Reshef, B.T. Nolan, J. Rubin and Y. Shaham-Gafni, "Model traceability," IBM Systems Journal, Vol. 45, No. 3, pp. 515-526, 2006.

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

[3] P. Bernstein, "Applying model management to classical meta data problems," In Proc. Conf. on Innovative Database Research, pp. 209-220, 2003.

[4] J. M. Favre, "Modelling and Etymology," in Transformation Techniques in Software Engineering, J. R. Cordy, R. Lämmel and A. Winter, Eds. Dagstuhl:IBFI, 2006.

[5] J. A. Goguen and R. M. Burstall, "Institutions: abstract model theory for specification and programming," J. ACM 39(1): 95-146, 1992.

[6] D. Jackson, Software Abstractions: logic, language and analysis, MIT Press, Cambridge, MA, 2006.

[7] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason IV, G. Nordstrom, J. Sprinkle and P. Volgyesi, "The generic modeling environment," Workshop on Intelligent Signal Processing, May 17, 2001.

[8] 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).

[9] B. Nuseibeh, J. Kramer and A. Finkelstein, "A framework for expressing the relationships between multiple views in requirements specifications," IEEE TSE, 20(10), Oct 1994, pp. 760-773.

[10] M. Sabetzadeh and S. Easterbrook, "An algebraic framework for merging incomplete and inconsistent views," 13th IEEE RE Conference, Paris, France, 2005.

[11] R. Salay, M. Chechik, S. Easterbrook, Z. Diskin, P. McCormick, S. Nejati, M. Sabetzadeh and P. Viriyakattiyaporn, "an Eclipse-based tool framework for software model management," ETX 2007 at OOPSLA, Oct. 2007.

[12] R. Salay, "Macro support for modeling in software engineering," Technical Report, University of Toronto. http://www.cs.toronto.edu/~rsalay/tr/macrosupport.pdf

[13] Software Process Engineering Metamodel V1.1. Object Management Group.