Simplifying OMG MOF-based Metamodeling

Nicolas F. Rouquette

Jet Propulsion Laboratory, California Institute of Technology

Abstract. What is a metamodel? "A metamodel is a model used to model modeling itself." Although this definition from the Meta-Object Facility (MOF) Core 2.5 specification lacks clarity, it is reflective of a cultural mindset about the relationship between models and metamodels. This paper proposes a different perspective about this relationship; briefly, equating a metamodel for a modeling language to the specification of a fourth normal form relational schema for the abstract syntax of that language and equating a model written in a modeling language to tabular data conforming to the relational schema of that language.
The essence of defining a fourth-normal relational schema for metamodel involves three kinds of tables: entity tables for each metaclass, directed binary relational tables for each association among metaclasses and attribute tables for each datatyped property. With a strict separation of modeling concerns amongst entities, relations and attributes tables, it is possible to explain traditional MOF-based metamodeling in a simpler way where metaclasses correspond to entity tables and associations correspond to relation tables as if they were directed and owned both of their association end properties. More importantly, this paper explains that this paradigm shift brings three tangible benefits for the practice of metamodeling at large: First, in lieu of a reflexive, multi-level metamodeling architecture like MOF where the notion of a profile seems to straddle between the metamodel and model levels, this paper explains a simpler paradigm from the perspective of equating a metamodel to the ontology of a terminology vocabulary and of using ontology terminology refinement as the basis for multi-level (meta) modeling. Second, in lieu of a complex model interchange paradigm based on serializing models using the XML Metadata Interchange (XMI) standard, this paper explains the benefits of serializing models in terms of their irreducible content as normal form relational data. Finally, this paper explains that the focus on normalizing the information schema of a model works in tandem with leveraging recent advances in functional programming languages like Scala to modernize the traditional practice of model-based programming with the Object Constraint Language (OCL) and the Query/View/Transformation (QVT) standards.

1 Contributions

This paper makes three significant contributions towards addressing significant issues with the OMG UML 2.5 specification:

1. Define a tool-neutral Application Program Interface (API) adaptable to tools claiming abstract syntax conformance to OMG UML 2.5.
   Unfortunately, the OMG stopped the UML 1.x practice of publishing revisions of UML 2 with an "Interface Definition Language" (IDL) API. Consequently, the OMG UML 2.5 specification lacks an objective basis for assessing whether "a tool demonstrating abstract syntax conformance provides a user interface and/or API that enables instances of concrete UML metaclasses to be created, read, updated, and deleted."
   This paper contributes a novel approach for generating and specifying a strongly type-safe abstract syntax API for OMG UML 2.5 parameterized by a tool-specific adapter for an existing implementation of UML 2.5. This API has been adapted for two existing UML 2.5 tools: NoMagic's MagicDraw version 18.0 and Eclipse UML version 5.0. The paper describes an approach for generating a suite of abstract syntax conformance tests that cold be used for objectively assessing the compliance of OMG UML 2.5 tool implementations and for objectively assessing model interchange via tool-specific API adapters instead of via external serialization as done before.
2. Define a set of 4th normal form database tables and relationships enabling a radically simpler tabular serialization of UML models compared to the complex tree serialization of UML models based on OMG's XMI 2.5 specification.
   For serialization, the OMG chose to represent the abstract syntax containment organization of a UML model as an XML tree. This seems a sensible choice given that a tree precisely captures the key architecture principle of single element ownership that applies to all OMG modeling languages, including UML. The complexity arises from two factors: 1) logically, a given tuple of owning element and owned element can be a link instance of multiple composite associations defined in the UML metamodel and 2) logically, the lack of clarity in the criteria for determining which composite association requires a serialization as an XML nested element from others requring serialization as an XML element reference.
   From a database perspective, the abstract syntax containment organization of a UML model is a materialized view of the information. The UML metamodel (a materialized view) has information redundancy that induces serialization redundancy: for example, every class-owned association end in the UML metamodel is both an attribute of its owning class and a member end of its association. Much of the complexity in OMG's XMI specification is an accidental consequence of trying to avoid unecessary redundant serialization. Normalizing the UML metamodel removes all information redundancy: for example, every association owns both ends. This paper presents a 4th-normal form of a UML metamodel where every metaclass is reduced to a table with only data property attributes and every association is reduced to a relationship table with an ordering data property attribute if one association end property is ordered. For a large model of a space mission (over 0.5 million UML elements), normal form serialization is considerably faster than the native tool-specific XMI tree serialization.

3. Define a description logic formalization of UML's abstract syntax for reasoning about UML syntax precisely instead of informally per the UML specification.
   UML provides a rich vocabulary for describing the structure and behavior of arbitrary systems. The UML vocabulary is extensible to accomodate domain-specific language modeling needs with the concept of UML profile extensions of the UML metamodel such as the Systems Modeling Language (SysML) for the domain of systems engineering. A typical concern in the development of a UML profile extension of UML/SysML for a domain-specific Model-Based Systems Engineering (MBSE) methodology is whether the intended expressiveness of the extended vocabulary in the specific domain can be represented as a well-formed UML/SysML model with the domain-specific profile applied. A serious concern is the limitations from the UML abstract syntax that prevent using the UML vocabulary for reasonable domain-specific modeling purposes. For example, the 6 of the 12 significant issues fixed in the OMG SysML 1.4 revision provide support for explicitly representing in the abstract syntax notions of scope and context that are currently implicit in the UML 2.5 abstract syntax. Unfortunately, the SysML 1.4 specification does not address the logical implications for managing the SysML extensions when the underlying UML elements are created, read, updated or deleted. This paper presents a simple strategy for leveraging a 4th normal form definition of the abstract syntax of UML and profile extensions such as SysML as the basis for ontological reasoning about extended UML models.

Acknowledgements

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

The author expresses gratitude to many colleagues at the Jet Propulsion Laboratory's Integrated Model-Centric Engineering project and the Laboratory for Reliable Software, in particular, M. Elaasar, K. Havelind, S. Herzig, S. Jenkins and R. Kumar, and to many current and past task force colleagues at the Object Management Group (OMG), in particular, Y. Bernard, C. Bock, R. Burkhart, S. Cook, S. Friedenthal, M. Elaasar, M. Koethe, P. Rivett, E. Seidewitz and B. Selic.