Simplifying OMG MOF-based Metamodeling

Dr. 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 MOF-based metamodeling is too complex

Several factors contribute to the complexity of MOF as an incomplete architecture for metamodeling and modeling with poorly separated concerns. The most significant factor pertains to the fundamental modeling constructs in the MOF

architecture, alasses, associations and properties; and more specifically to their lack of orthogonality since an association can be modeled as a class with a pair of properties. In practice, poor separation of concerns between entity and relationship constructs invites significant conceptual redundancy and variability in their use. For example, a relationship among entities may be modeled via several patterns of fundamental constructs: as an association (with association end properties), as properties of the related entities (without an association), as a (relationship) class with properties or as even more redundant combinations of the above. Since the Object Management Group (OMG) uses MOF for defining metamodels, construct redundancy affects all OMG metamodels like the Unified Modeling Language (UML) where redundancy leads to significant variability in the way conceptual relationships are defined in the UML metamodel.

The UML metamodel is defined in terms of MOF classes, associations and properties. A MOF class in a metamodel for a modeling langauge represents a concept in that language. There is no terminology consensus for what MOF associations and properties represent in the modeling language. The metaclasses in the UML metamodel are organized in a classification taxonomy with a single root metaclass, UML Element, which is the toplevel concept in UML for a constituent of a model. All the other metaclasses in the UML metamodel are directly or indirectly classified as a kind of UML Element. For example, a UML Relationship is a kind of UML Element specifying some kind of relationship between other UML Elements. Unfortunately, The UML Relationship metaclass does not classify all of the UML metaclasses that conceptually represent some kind of relationship in the language, for example:

1. In the graph of a UML Activity, the ActivityEdge metaclass represents a directed relationship from an ActivityNode to another.
2. In the graph of a UML Interaction, the GeneralOrdering metaclass represents a directed relationship from an OccurrenceSpecification to another.
3. In the graph of a UML Interaction, the Message metaclass represents a trace relationship between send and receive events.
4. In the graph of a UML StateMachine, the Transition metaclass represents a directed relationship from a Vertex to another.
5. In the graph of a UML StructuredClassifier, the Connector metaclass represents a join relationship among ConnectorEnds.

In all five examples, the conceptual relationship is represented as a metaclass; however, these examples differ in the way related entities are represented. For ActivityEdge (1) and GeneralOrdering (2), each related entity is represented as an undirected association with non-derived class-owned association end properties. This is done differently for the other cases (3,4,5): The representation of the source and target vertices of a Transition (4) is closest to (1) and (2) except that the association end property typed by the relationship itself is derived in (4) compared to (1) and (2) where it isn't. The representation of the send and receive events of a Message (3) involves a directed association from the relationship metaclass (Message) to the related entity metaclass (MessageEnd). This is in contrast to (1,2,4) where the association is undirected. Finally, the

representation of the related ConnectorEnds for a Connector (5) involves a single undiferentiated directed composite association unlike all other cases (1,2,3,4) where the relating association is non-composite. Although there are legitimate syntactic concerns behind the variability of these five cases, the lack of syntactic homogeneity in the way conceptual relationships are represented in the UML constitutes a significant source of metamodeling complexity.

This paper argues that the root cause of metamodeling complexity stems from a design choice to provide support for syntactically representing characteristics of conceptual relationships that have subtle interdependencies:

- The syntactic variability of association end property ownership (association-owned vs metaclass-owned) affects property subsetting and redefinition since the subsetting and redefinition contexts respectively are different in each case.
- Syntactically, property characteristics such as ownership, aggregation, derivation, subsetting and redefining are independent of each other. However, the syntactic coupling of properties as opposites of a binary association end forces subtle restrictions on well-formed syntactic variations. These subtilities were poorly understood from UML 2.0 until UML 2.3. A key goal of the MOF 2.4 and UML 2.4 revisions was to formalize the well-formedness constraints applicable to all MOF metamodels and to verify the well-formedness of UML 2.4. Achieving this goal involved hundreds of repairs; far too many to manually review so special tools were required to mechanically verify them. However, a subtle error slipped through all reviews that required an urgent fix and the publication of UML 2.4.1[1]:

  > When UML 2.4 was released we discovered an issue that meant it was impossible to interchange StructuredActivityNodes reliably. StructuredActivityNodes are executable nodes in an activity diagram that are also groups. UML 2.4 doesn't specify clearly whether they should be serialized as nodes or as groups; as a consequence, different tools do different things. In UML 2.4.1 this ambiguity is remedied by serializing them in their own collection. UML 2.4.1 can be found at http://www.omg.org/spec/UML/2.4.1/ and all of the machine-readable files are at http://www.omg.org/spec/UML/20110701/.
  > This is the first time in the history of UML that a complete machine-readable definition of the language is available online, defined as an instance of itself, and with no dangling references. Only of interest to UML aficionados maybe, but a milestone nonetheless.

Surprisingly, although the OMG positioned MOF as an architecture for defining and managing metadata information, there is no evidence in the OMG specifications to suggest that the proven principles of information modeling with the relational model were taken into consideration, if at all. This paper departs from the traditional object-oriented modeling & programming perspective that influenced

[1] https://blogs.msdn.microsoft.com/stevecook/2011/10/24/uml-2-4-1-now-released/

the original definition of UML1 and the major revision in UML2. Notwithstanding considerations of normalization in relational modeling as a pragmatic criteria for simplifying the current MOF paradigm, there is plenty of historical evidence in the records of OMG task forces about the difficulties encountered in understanding the particular patterns of fundamental constructs used in specifying a metamodel or a profile and in correcting errors due to differences between the intended and the actual meaning of the abstract syntax according to the OMG task force vs. the patterns of MOF fundamental constructs used. Historically, the cost of finding and correcting these discrepancies has been very high in terms of the man/years of effort by OMG task forces and by tool vendors implementing revisions of OMG specifications. Experience suggests that the current OMG processes for revising OMG specifications incur significant missed opportunity costs because of the lack of pragmatic rigor in exploiting modern computer science techniques for rigorous specification development, for example:

- Formal methods help ensure that a system behaves according to its specification.
  How about verifying that a system (e.g. an implementation of a modeling language like the UML) behaves according to its specification (e.g., the OMG UML abstract syntax metamodel)?
- *A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute[1].*
  The OMG publishes several modeling specifications with executable semantics such as Alf & fUML and BPMN. Alf is an example of a recently developed specification at the OMG where the Alf modeling language is specified in the same fashion as programming languages are, that is, with an explicitly defined type system. Alf is an exception at the OMG.

In particular, it is noteworthy to emphasize that the OMG publication process requires every OMG modeling specification to specify criteria of conformance to the specification. Such criteria pertain to notions of abstract and concrete syntax, model interchange, diagram interchange and semantics. Historically, this process requirement has received very little attention from the computer science community. In particular, even though every OMG modeling specification includes some kind of abstract syntax conformance criteria, none do so in terms of a specification for the program interface of a system implemention. This paper contributes to bridging this gap on two levels:

1. Defining an Application Program Interface (API) for a simplified OMG's MOF 2.5 based on a normalized relational model.
2. Generating an Application Program Interface (API) from the simplified MOF abstract syntax metamodel of OMG UML 2.5.

The first step towards bridging the gap between the OMG and computer science culture is simplifying OMG's MOF 2.5

2 An ontological normal form of a MOF

This section explains the rationale of each step involved in simplifying OMG's MOF into a irreducible, ontological, normal form information schema.

2.1 Which of the three concepts is redundant?

Having established the redundancy of MOF Class, MOF Association and MOF Property for defining metamodels, which of these three can be eliminated? MOF Class is a first-class concept in metamodels because every element in a model must be an instance of at least one (meta)class defined in the metamodel that the model conforms to. Historically, there's been much debated about which of MOF Property and of MOF Association needs first-class status. MOF Property is a first-class concept in the Eclipse Modeling Framework (EMF). However, full support for binary MOF Associations requires a suitable EMF code generator represent them in EMF without loss of information. In terms of OMG's MOF 2.5 and EMF 2.12, there are three variations of binary MOF Associations to consider: 1) metaclass-owned association end properties; 2) one association-owned end and one metaclass-owned end; 3) association-owned ends. EMF directly supports case (1) only: metaclass-owned association end properties are represented as opposite EMF `EReferences` to update the opposite when one end is updated. The other two cases depend on the EMF code generator used. For example, the Eclipse UML code generator adds to each metaclass that is the type of an association-owned end property an EMF annotation that effectively acts as if the association-owned end property were instead owned by that metaclass. Since full association support depends on EMF code generation techniques, navigating models using the EMF `EObject` and `EReference` API is limited to the first case only. For cases (2,3), API-based navigation requires knowledge of the code generation encoding of association-owned ends, if they are represented at all[2] Since EMF is widely accepted as the de-facto open-source reference implementation of OMG's Essential MOF (EMOF) subset[2, section 2.6.2], the above analysis should suffice to claim that EMF is insufficient for code-generation agnostic API-based navigation of models according to all three cases of Complete MOF (CMOF) binary associations. Therefore, one must conclude that class+association are first-class concepts and that property isn't.

2.2 Deconstructing the concept of CMOF property

MOF Property is not a first-class concept because it lacks conceptual unity. Indeed, a property can play different roles in a CMOF metamodel. In the absence of an official terminology, the terms used in this paper are underlined:

1. A metaclass or datatype attribute.

[2] The default EMF code generator does not map association-owned end properties[2, See section 6.4].

This case is outside the scope of the fundamental constructs of metamodels. An important simplification of current OMG practices is a clean separation of datatype libraries as a kind of model that can be used in other kinds of non-library models, i.e. metamodels, profiles and models that are instances of metamodels possibly extended with profiles. Clearly, metamodels and profiles need support for datatyped attributes; this is a minor technicality that will not be further elaborated in this paper.

2. A metaclass property.

   (`type` is a metaclass)

   In principle, CMOF constraints allow a metaclass to own a non-association end property typed by a metaclass. Without loss of generality, this paper considers this case to be a degenerate of the next case and can be refactored accordingly by explicitly defining an association.

3. A binary association end.

   (`type` is a metaclass; `aggregation=none|composite`; `isID=false`)

   Since ends are an essential part of the definition of an association, the concept of MOF binary association is augmented to include the relevant characteristics of both ends (i.e., `type`, `lowerValue`, `upperValue`, `isOrdered`, `isUnique`, `isDerived`, `isDerivedUnion`, `isReadOnly`, `aggregation`, `subsettedProperty`, and `redefinedProperty`). Note that the last three characteristics are coupled between the two ends[3]:

   - `aggregation`: Only one end may be `composite`.
   - `subsettedProperty`: must be symmetric.
   - `redefinedProperty`: must be accompanied by corresponding subsetting or redefinition at the other end.

   Symmetric subsetting means that link instances of an association with subsetting ends must be also link instances of the other associations with the subsetted ends. This implies that an association with subsetting ends effectively specializes the associations whose ends are subsetted. Like subsetting, redefinition has a semantics of association specialization but with an additional forcing semantics in the contexts of the redefining ends: In such contexts, the redefining ends replace the redefined ends. This means that in such contexts, it is not possible to create link instances of the associations with redefined ends because such links must be instead instances of the association with redefining ends.

The rest of this paper focuses only on case (3); the role of a property as an end of a binary association in a metamodel.

[3] See resolutions of issues 14993 and 14977 in UML 2.4.1[4] for the last two.

2.3 Simplifying non-union derived association end properties

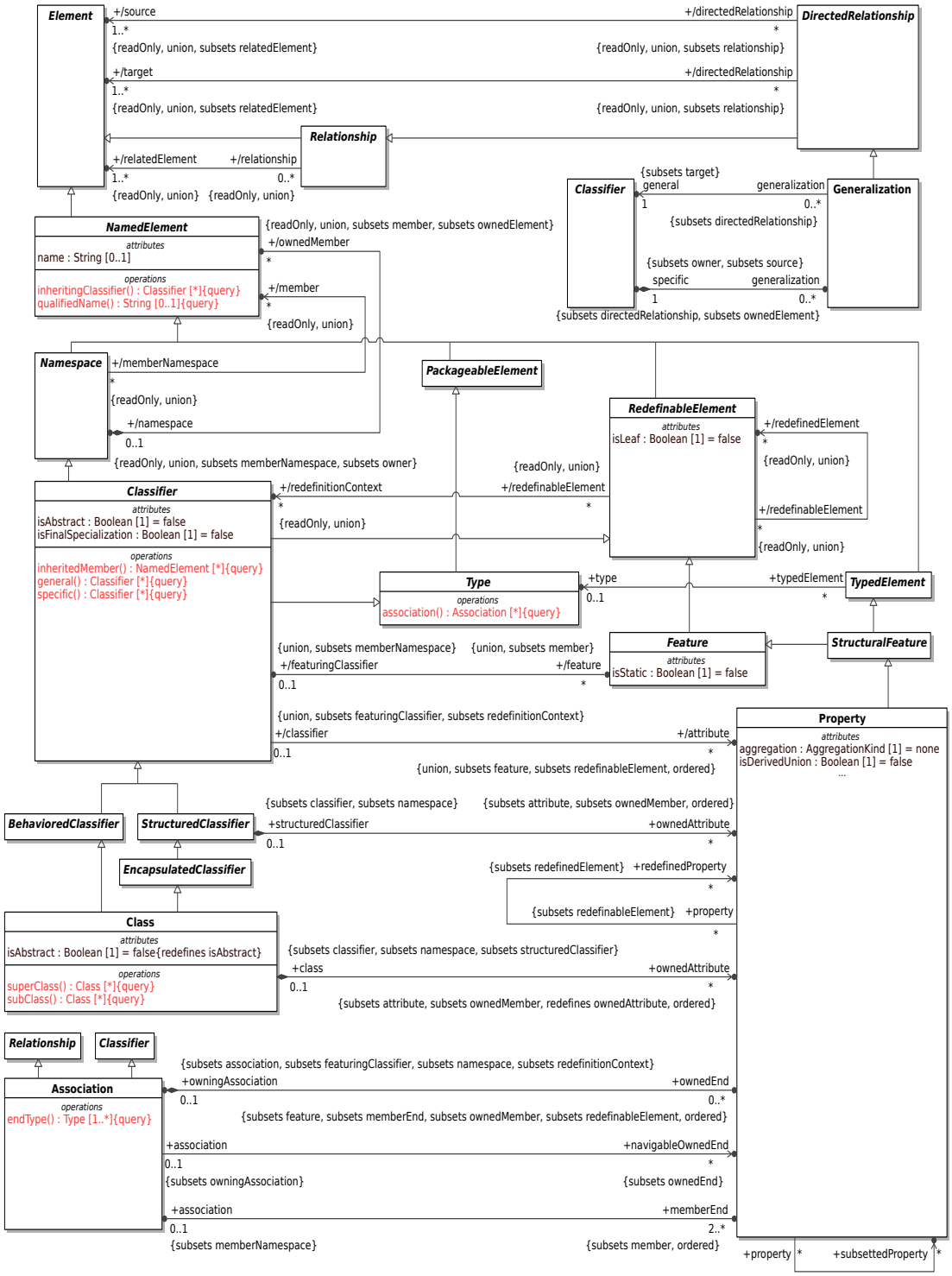

Fig. 1. Simplified CMOF metamodel.

The UML metamodel adopted a convention where non-union derived association end properties have a corresponding operation query[5, section 6.4.1]. The reckoning of this redundant representation is that the operation query enables specifying the OCL rule while the association end property enables specifying the availability of the derived property. This redundant modeling reflects historical practices that result in unecessary confusion and complexity. Often, only the metaclass-owned end is derived, the association-owned end isn't. This creates an unecessary source of confusion. Since nagivation is irrelevant for the purposes of OCL, it makes sense to treat such associations as if both ends are metaclass-owned and derived. Then, instead of duplicating each derived end with a corresponding operation query, it makes sense to delete the association altogether. Figure 1 shows the result of carrying out these simplification steps to the subset of the UML 2.5 metamodel corresponding to the fundamental constructs in CMOF metamodels (i.e. class, association and association end property). Datatyped attribute properties are not shown as explained previously. The operation queries corresponding to deleted associations with derived ends are shown in red. Several metaclass attributes are no longer necessary and were deleted (Association::isDerived, Property::isDerived, StructuralFeature::isReadOnly)

2.4 Deconstructing the concept of CMOF association

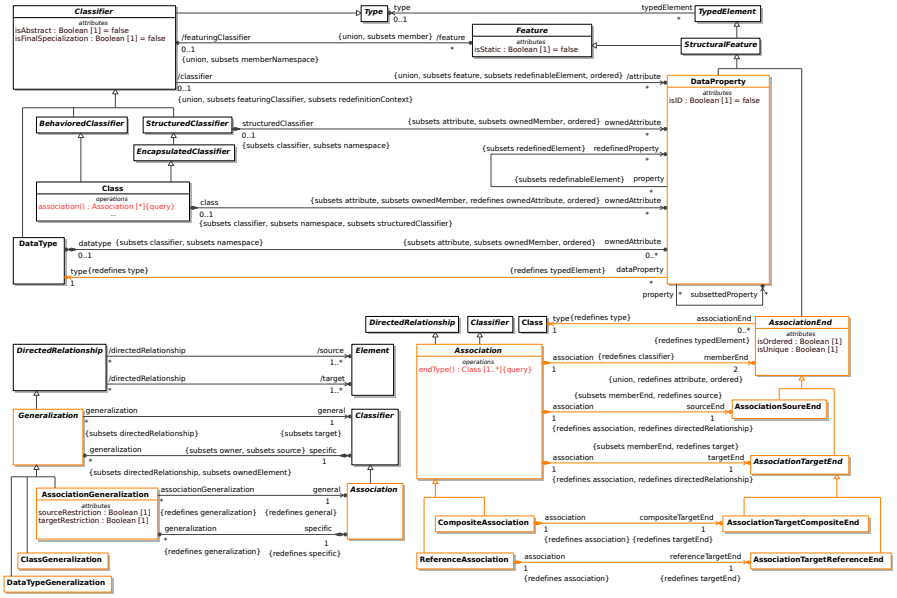

Fig. 2. Refactored CMOF metamodel.

The key to understanding the simplification of CMOF binary association and of association end properties is to recognize that an association end property specifies two distinct aspects that are currently convolved in the concept of CMOF property: 1) a collection of values and 2) an operation for querying this collection. Unfortunate historical circumstances and a misplaced preoccupation with object-oriented programming contributed to muddling these two aspects under the bloated concept of CMOF property. The semantics of a property as a collection of values is explained in UML 2.5. The semantics of a property as a query operation has been an important topic in the major revision of OCL 2.0, so much so that navigating across non-navigable ends has been a compliance point ever since. Unfortunately, OCL 2.0 was not ready when the OMG adopted major architecture changes from UML 1.x and MOF 1.x to UML 2.0 and MOF 2.0 in 2003, changes that were finalized in 2005 a year before OCL 2.0 in 2006. the major architecture changes from UML 1.x and MOF 1.x to UML 2.0 and MOF 2.0 in 2003 and finalized in 2005. Perhaps the influence of the object-oriented programming paradigm from the simpler Essential MOF with classes and properties contributed to muddling these aspects. Indeed, avoiding strong references from metaclass-owned association end properties was historically used as the reason for making association ends owned by the association. It took several years until the publication of FUML 1.0 in 2010 before the emphasis was made on the fact that, at the model level, values of association end properties are set, cleared and updated not individually but rather via actions on link instances of associations. Unfortunately, even FUML 1.2.1 does provide symmetric support for instances of metaclasses and of associations as suggested by the MOF abstract semantics. Indeed, FUML restricts the ReadExtentAction to classifiers that are classes, not associations.

Separating collection from query aspect enables a significant simplification of CMOF, particularly since the query aspects can be fully generated from the collection aspects like Eclipse EMF code generation does for metamodel API and implementation. Thus, the refactored CMOF metamodel shown in 2 separates the roles of a CMOF property as discussed in Sec. 2.2: DataProperty and AssociationEnd correspond respectively to cases 1 and 3. Since AssociationEnds are always attribute features of their owning Association because they cannot be members of two distinct namespaces like class-owned properties are in UML. Albeit a poorly understood consequence of UML 2.5 & MOF 2.5, a CMOF metamodel binary association is always directed from a source end to a target end according to a prioritized list of six criteria:

1. The target end is composite, the source end is not.
2. The source end is owned by the association, the target end by a metaclass.
3. Both ends are owned by the association, the source end is navigable, the target end isn't.
4. Both ends are not composite, the source end is unbounded, the target end has a finite upper bound.
5. The source end (resp. the target end) directly or indirectly subsets or redefines another source end (resp. target end).

6. The source and target ends are respectively the first and second properties in the ordered member end collection.

Instead of carrying this archaic criteria, the refactored CMOF metamodel explicitly differentiates at the metaclass level source and target association ends. Aggregation is only relevant for a target end property. Additional simplification stems from reflecting in the abstract syntax the difference that composite vs. reference (i.e. aggregation=none in UML2.5 terminology) makes for the semantics of associations. Finally, since generalization has different semantics for the kind of classifier involved, it is consequently specialized in the abstract syntax. This enables replacing the complex UML2.5 notions of property subsets and redefinitions with a much simpler boolean flag corresponding to the semantics of symmetric subsetting and of redefinition forcing: the source (resp. target) restriction flag on AssociationGeneralization indicates whether the generalization corresponds to the weak case of subsetting (i.e., restriction = false; the value of the specialized end is existentially constrained to be a subset of the value of the general end) or to the strong case of redefinition with forcing semantics (i.e., restriction = true; the value of the specialized end universally restricts the value of the general end).

3 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 def-

inition 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.

References

1. B. C. Pierce, "Types and Programming Languages", 2002.
2. D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, "EMF Eclipse Modeling Framework", 2nd edition, Addison-Wesley, 2008.
3. Object Management Group, "Meta-Object Facility Core Specification", version 2.5, formal/2015-06-05, 2015.
4. Object Management Group, "Report of the UML version 2.4.1 Revision Task Force", ptc/2011-01-19, 2010.
5. Object Management Group, "Unified Modeling Language version 2.5", formal/2015-03-01, 2015.