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 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[3].*
  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 Simplifying 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[4, 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 <u>attribute</u>.

[2] The default EMF code generator does not map association-owned end properties[4, See section 6.4].

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.

2. 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.
3. 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 previous case that can be refactored accordingly by explicitly defining an association.

2.3 Simplifying non-union derived association end properties

The UML metamodel adopted a convention where non-union derived association end properties have a corresponding operation query[9, section 6.4.1]. Historically, this redundancy was rationalized on the basis that the operation query enables specifying the OCL rule while the association end property enables specifying the availability of the derived property. This redundancy creates unecessary confusion where a derived association has a derived metaclass-owned but a non-derived association-owned end! 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

[3] See resolutions of issues 14993 and 14977 in UML 2.4.1[8] for the last two.

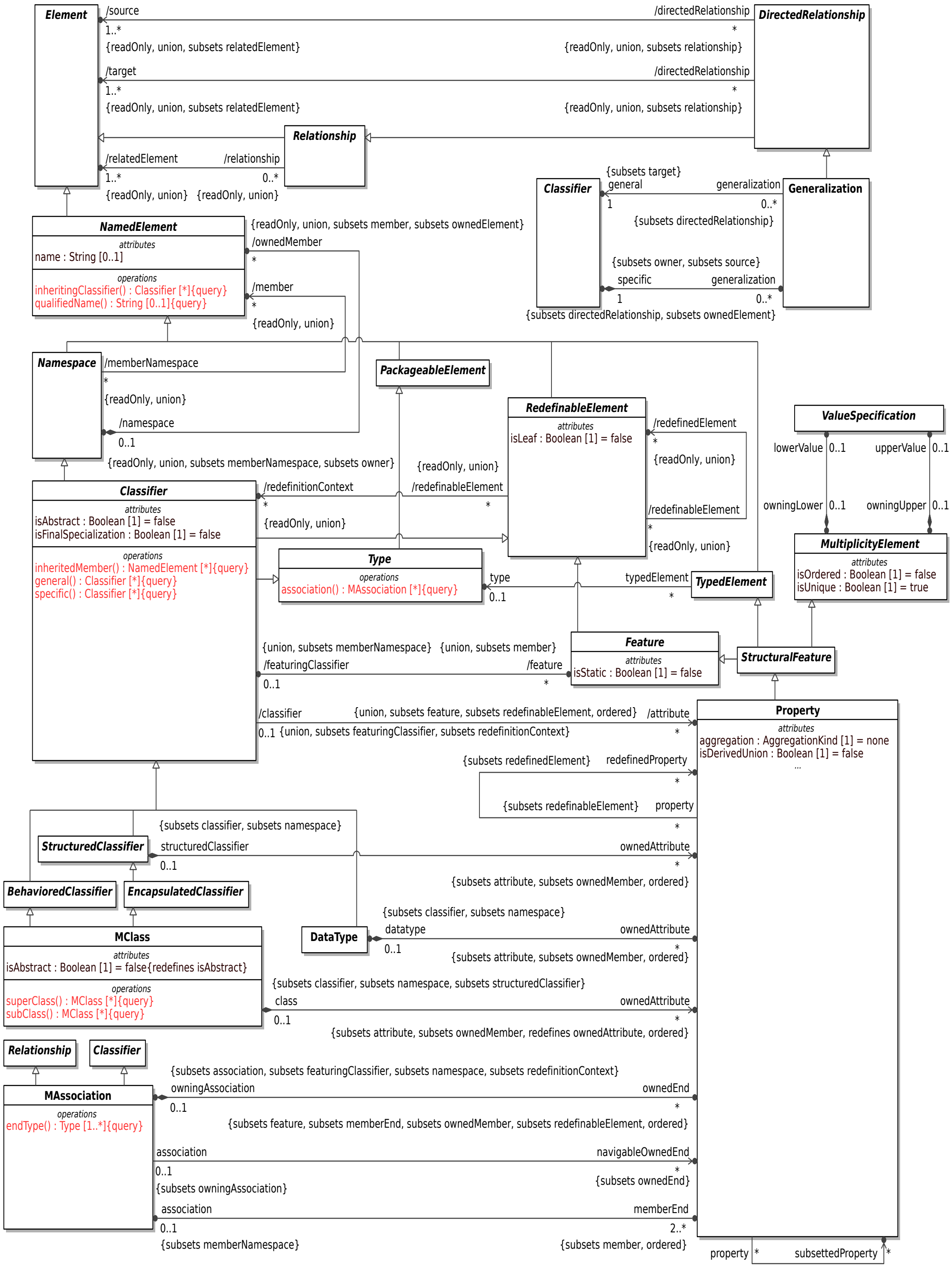

Fig. 1. Simplified CMOF metamodel.

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 fundamental constructs including the two variants of properties described previously.

Whereas the OMG emphasizes the circular definition of UML as a CMOF metamodel which is itself defined as a subset of UML, this paper claims that this is another unecessary source of complexity. To emphasize the strict separation of levels between the CMOF metamodel itself vs. CMOF models (e.g. the UML itself), Class and Association are prefixed 'M' (Meta) to differentiate these CMOF metamodeling constructs from similarly named constructs in the CMOF model of UML. Operation queries corresponding to the ends of deleted derived associations 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

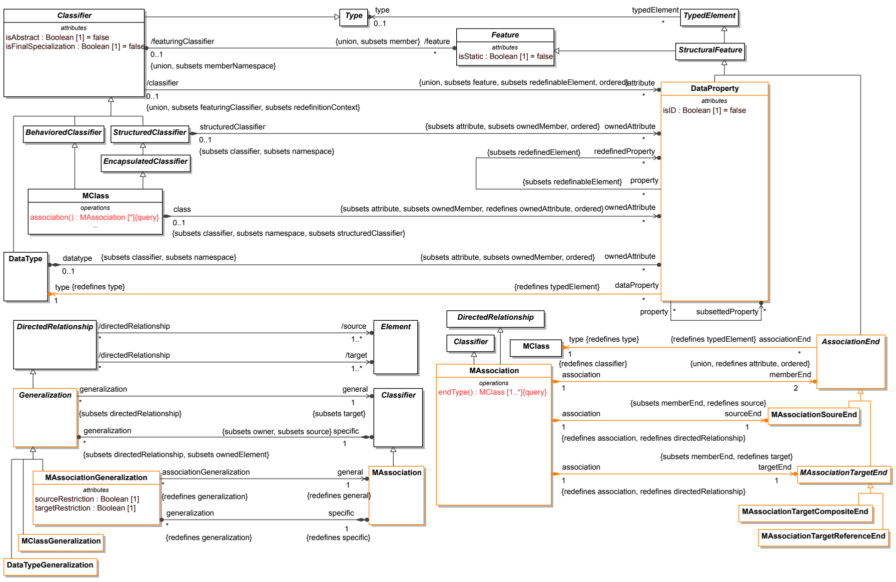

Fig. 2. Refactored CMOF metamodel.

Although the concept of association is a first-class construct for defining metamodels, the OMG made significant changes to the concept of association end property. The UML 1.x and MOF 1.x specifications used the term 'property' in a general sense for any of three distinct kinds of metaclasses: AssociationEnd (i.e., 2), Attribute (i.e., 1) and TaggedValue (not described in this paper). As part of the major 2.0 revision, these three metaclasses were replaced with a single one, Property. Although a few 1.x distinctions were encoded as MOF constraints, the major revision lost the 1.x separation between AssociationEnd and Attribute. The 2.0 revision added support for the object-oriented paradigm where an association end Property can be owned by a Class or by an Association. Class-owned association end Properties enabled support for Essential MOF metamodels where the semantics of a binary Association is defined in terms of its Class-owned Properties only [4, See section 6.4]. Historically, the 2.0 major revisions of UML and MOF were adopted in 2003 and finalized in 2005; at least a year before the finalization of the 2.0 major revision of OCL was finalized in 2006. OCL 2.0 emphasized the operational aspects of querying association end properties regardless of their ownership by a class or association. This emphasis is so important for OCL that it has been a compliance point ever since. OCL's emphasis on the query semantics of navigating association end properties stands in contrast with UML's emphasis on the value semantics of association end properties and on the class/property object oriented view of associations. Later in 2010, the Foundational subset for executable UML (FUML) specification emphasized the lifecycle aspects of creating and deleting link instances of associations based on the existing UML action semantics. FUML in fact requires associations to own all of their ends but unlike the misconception from the object-oriented paradigm, supports operating on association end properties from related classes as if such ends were class-owned features. However, FUML does not give association links a similar status as that of class instances because FUML restricts the classifier queried by a UML ReadExtentAction to be a class only. The rest of this section explains the basis for simplifying CMOF meta associations in terms of the three aspects of association end properties that have historically been the source of much complexity, confusion and errors in OMG specifications: aggregation, ordering and subsetting/redefinition.

Simplifying and promoting aggregation Since Property::aggregation has a significant effect on the semantics of association end properties, the difference is elevated to the conceptual level instead of being represented in terms of the Property::aggregation attribute as in current MOF, UML and FUML. Hence, the refactoring of the relevant subset of the CMOF abstract syntax metamodel shown in Fig. 2 distinguishes the roles of a CMOF property with respect to typing like UML 1.x and MOF 1.x did (DataProperty corresponds to case 1 and AssociationEnd to case 2) and, in the latter case, further distinguishes the roles of CMOF MetaAssociation and of MetaAssociationEnd with respect to aggregation (this distinction has no counterpart in past & current OMG specifications). Like UML 1.x and FUML, association ends are semantically owned features of their association.

Simplifying association end ordering Although association ends have always been ordered in UML 1.x and 2.x, this syntactic ordering is independent from all the other characteristics of association ends (aggregation, ownership, navigability and multiplicity). In practice, such characteristics are typically used to explain the intended ordering instead of using the notation for the association end ordering (see [9, Section 11.5.4]. The refactored CMOF metamodel is designed to be compatible with current metamodeling practices, in particular, it reflects the practice of infering association end ordering from their characteristics according to the following prioritized 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. If none of the above applies, the source and target ends are respectively the first and second properties in the ordered member end collection.

Instead of carrying such complex criteria, the refactored CMOF metamodel explicitly differentiates at the metaclass level the source and target association ends. That is, the ordering of the association ends for a current MOF 2.5 binary association must be determined according to the six criteria above whereas in the refactored CMOF metamodel, the ordering is explicitly represented in differentiated metaclasses, i.e. MetaAssociationSourceEnd and MetaAssociationTargetEnd. Additional simplifications were possible thanks to the asymmetry of aggregation, which is only relevant for an association target end. Since aggregation has profound semantic implications for the lifecycle semantics of classifiers and their instances, it is an essential characteritic distinguishing MetaAssociationTarget{Composite,Reference}End.

Simplifying subsetting and redefinition After the UML 2.0 major revision, the relationship between association specialization and association end subsetting or redefinition were poorly understood. This topic was the subject of intense scrutiny in the UML 2.4.1 revision because hundreds of errors were traced to inconsistent and/or incorrect subsets and/or redefinitions. Unfortunately, a clear explanation is missing from the UML 2.5 simplification. Here, the refactored CMOF metamodel reflects a unified ontological view of meta associations as relationships that classify related metaclasses in terms of named roles. In this ontological view, subsetting and redefinition have the semantics of restricting the subsetted or redefined association respectively. More precisely, AssociationGeneralization with source and target restriction=false corresponds to existential subsetting; that is, the weak restriction that a link classified by the specialized (i.e. subsetting) association must be some link classified by the general (i.e. subsetted) association. AssociationGeneralization with source or target restriction=true corresponds to a universal redefinition; that is, the strong restriction that all links classified by the general association (i.e., with the redefined source and/or target end) must also be classified by the specific association (i.e. with the redefining source and/or target end.)

3 A normalized relational schema for ontological resources

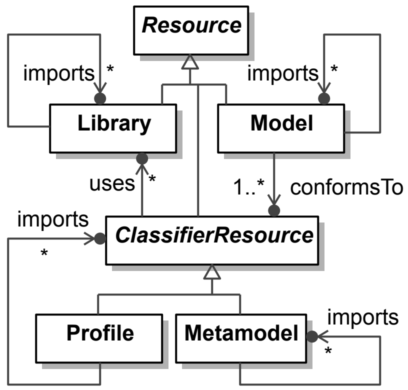

Fig. 3. Ontological Resources

From a relational modeling perspective, the refactored CMOF metamodel shown in Fig. 2 would be a materialized view a normalized relational model for metamodeling that has never been defined at the OMG. This section describes normalized relational models for four semantically disjoint categories of resources shown in Fig.␣3: libraries (Fig. 4), metamodels (Fig 5), profiles (Fig 6) and models (Fig. 7). An explicit concept of Resource is missing from the OMG specifications: Indeed, although MOF defines the concept of Extent (see [6, section 10.2]) and XMI defines the scope of XMI document serialization and deserialization in terms of the unspecified concept of "model or model fragment" (see [6, section 9.2]), the two notions are unrelated. Import relationships shown in Fig.␣3 correspond to UML PackageImport and ProfileApplication augmented with programming language-like semantics (cross-references from one resource to another are well-formed if and only if there is a corresponding import relationship) and kinding restrictions (e.g., importation is homogeneous for libraries, models and metamodels; profiles can import either profiles or metamodels). The kinding restrictions induce three layers of resources: 1) libraries that can acyclically import each other; 2) metamodels and profiles that can acyclically import each other and use libraries and 3) models that can acyclically import each other and that must conform to at least one metamodel or profile (ProfileApplication-like semantics) and transitively to any other resource directly or indirectly imported or used. Conformance for a model means that all of the elements, links and values in the model extent must be conforming instances of meta classes, meta associations, stereotypes and datatypes defined in the defined in the metamodels, profiles and libraries that the model directly or indirectly conforms to.

The following explains the notation used for the different kinds of tables in the normalized relational model shown in Figs. 4, 5, 6 and 7:

- An entity table shown in white carries identity criteria, at minimum, a primary key (uuid). Some entities have a name property as a secondary key in accordance to the UML namespace distinguishability contains. AssociationEnd also includes a ternary key, isOrdered, because it is an essential characteristic of the entity. An entity corresponds to a Sortal in OntoClean␣[10].
- An attribute table shown in yellow class defines a single attribute property typed by a primitive type and relates to a single entity. An attribute table corresponds to an Attribution in OntoClean␣[10].
- A relation table shown in bold gray has at least one foreign key. Some have an additional property typed by a primitive type (e.g., index, value). Together, the values of all foreign keys and attributes uniquely identify an instance of a relation table. Such a relation table corresponds to an optional MaterialRole in OntoClean␣[10] of the S
- A relation table shown in plain gray has at least two foreign keys. Some have an additional property typed by a primitive type (e.g., index). Together, the values of all foreign keys and attributes uniquely identify an instance of a relation table. Such a relation table corresponds to an essential FormalRole in OntoClean␣[10].

Note that compared with CMOF[6], the normalized relational model provides no support at all for any behavioral feature of any kind. This a deliberate design decision to separate the concerns of managing modeling resources in terms of relational data from the concerns of querying, transforming and reasoning about such resources in terms of functional programs operating on such resources.

3.1 A library is a resource of datatype classifiers

The MOF extent of a Library resource is exclusively the set of normalized entities, attributes and relationship tables defined in the Library package and those related to the DataTypedFeatures used from the Features package as shown Fig. 4. In contrast to OMG modeling specifications, including variants of MOF and UML, that allow defining datatypes anywhere and that result in significant duplication, the approach described here promotes a clean separation of concerns for entities and relationships pertaining to the definition of conceptual vs. data vocabularies. Conceptual vocabularies are the exclusive province of metamodels and profiles. Data vocabularies are the exclusive province of libraries. These two categories of vocabularies are seldom separated despite having fundamentally very different kinds of semantics: the semantics of conceptual vocabularies is about identified instances of sortal entities and of their relationships via formal roles whereas the semantics of data vocabularies is based on structural equivalence of structured datatype values and equality of atomic datatype values. Note that a StructuredValue carries an identity criteria, uuid. This is a deliberate choice for simplifying change management from the complexity of OMG's XMI tree-based serialization to the simplicity of adding/deleting rows for entity, attribute or relationship tables where a row is comprised of a tuple of key values (uuids) or lexical representations of values of atomic datatypes [12,

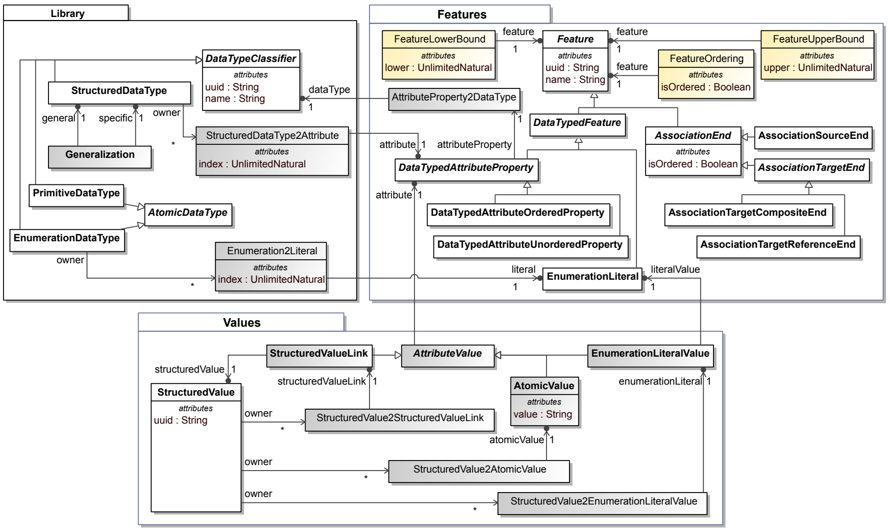

Fig. 4. Libraries

Section 2.3]. For example, the values of the attributes of a StructuredValue entity are specified via separate, essential formal role relations: Structured-Value2{StructuredValueLink, AtomicValue, EnumerationLiteralValue}.

3.2 A metamodel is a resource of meta-classes and associations

The MOF extent of a Metamodel resource is exclusively the set of normalized entities, attributes and relationship tables defined in the Metamodel package and those used from the Features package as shown in Fig. 5, which is considerably simpler than current CMOF and even the refactored CMOF shown in Fig. 2. Note that the extent of a Metamodel resource also includes all of the Features-based entities, attributes and relations involved in specifying the optional and essential roles of metaclasses and associations. However, the DataTypeClassifiers that are the dataTypes of MetaClass attributes must be directly or indirectly imported from libraries.

3.3 A profile is a resource of stereotypes

The MOF extent of a Profile resource is exclusively the set of normalized entities, attributes and relationship tables defined in the Profile package and those used from the Features package as shown in Fig. 6, which is considerably simpler than current UML Profiles (See [9, Section 12.3]). The simplification stems from eliminating associations among stereotypes because these add significant complexity for no demontrated practical value (See the example in UML 2.5

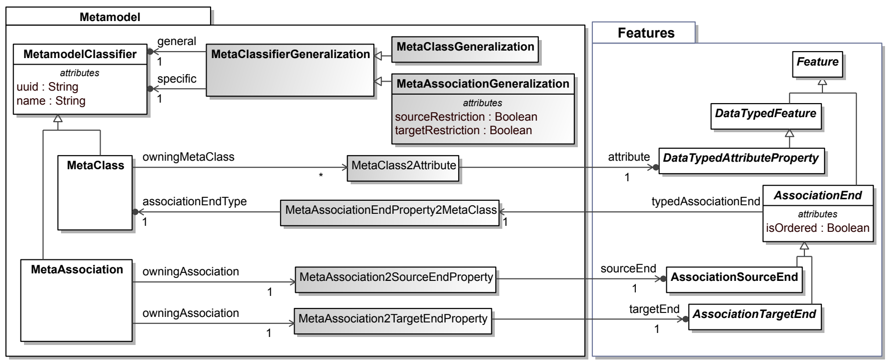

Fig. 5. Normalized Relational Schema for Metamodels

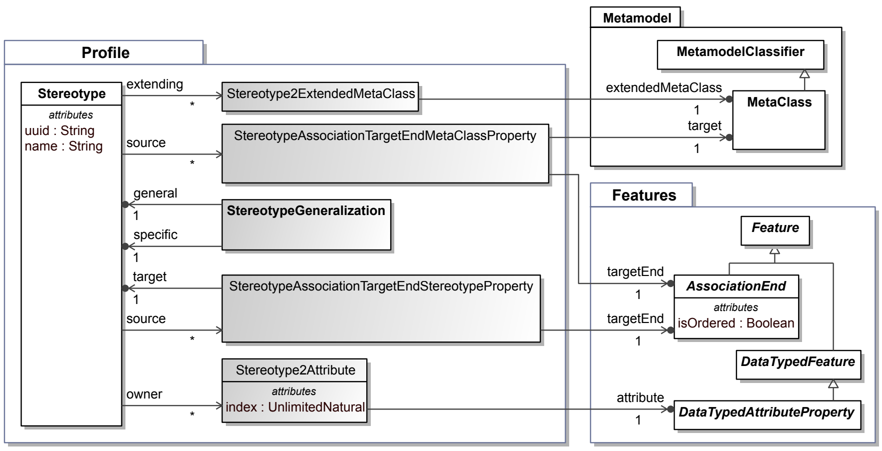

Fig. 6. Normalized Relational Schema for Profiles

[9, section 12.3.5]), retaining only the AssociationEnds corresponding to the so-called stereotype tag properties. Such AssociationEnds can play two distinct roles according to their type (MetaClass vs. Stereotype). Note that the extent of a Profile resource also includes all of the Features-based entities, attributes and relations involved in specifying the optional and essential roles of stereotypes. The meta classes referenced as targets of Stereotype2ExtendedMetaClass must be directly or indirectly imported. The stereotypes referenced as targets of StereotypeAssociationTargetEndStereotypeProperty or as general/specific of StereotypeGeneralization must be defined in the same resource or must be directly or indirectly imported. Similar to metamodels, the DataTypeClassifiers that are the dataTypes of Stereotype attributes must be directly or indirectly imported from libraries.

3.4 A model is a resource of model elements and links

The MOF extent of a Model resource is exclusively the set of normalized entities, attributes and relationship tables defined in the Model package and those used from the Values package as shown in Fig. 7, which is substantially simpler and more comprehensive than in current OMG specifications.

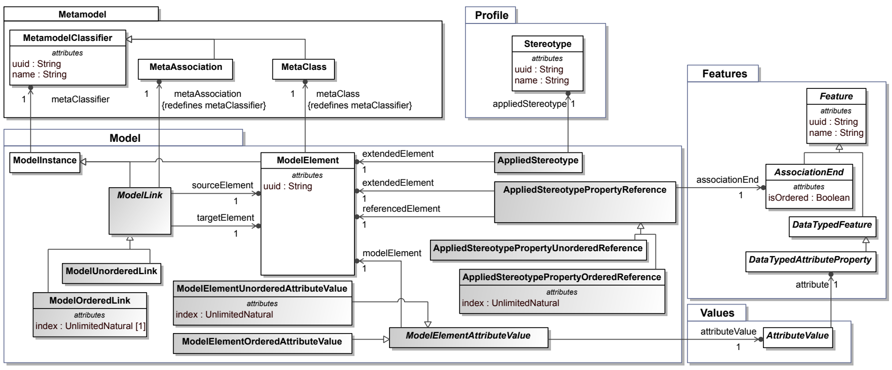

Fig. 7. Normalized Relational Schema for Models

As explained in CMOF Reflection[6, Section 13], class and association classify model elements and links respectively. In contrast to the subjective rationale in CMOF Instances Model [6, Section 15.3] for a redundant representation of links via CMOF AssociationInstances and CMOF Slots on linked CMOF ElementInstances for metaclass-owned association ends, this paper follows on the footsteps of FUML to unify and simplify CMOF abstract syntax and semantics: first-class classifiers (MetaClass and MetaAssociation) have corresponding first-class instances (ModelElement and ModelLink respectively).

In contrast to the UML 2.x specification which defines the abstract syntax for Profiles but not for their application, which is instead described via examples of XMI serialization, this paper defines an abstract syntax for stereotypes applied to elements and their so-called tag property values as shown in Fig. 7 based on a simplification of profile semantics compared to the CMOF-equivalent semantics described in UML 2.5 [9, section 12.3.3]: AppliedStereotype is an optional classification of a ModelElement as an instance of the Stereotype applied; that is, values of Stereotype attributes are represented with the same mechanism as are values of MetaClass attributes (i.e., ModelElementAttributeValue). This avoids the complexity of the CMOF-equivalent semantics of Extensions as Associations while retaining the intent of the CMOF-equivalent semantics of a Stereotype as a CMOF class that can be optionally applied to an element via the AppliedStereotype optional role.

ModelLink corresponds to the concept of link in OMG specifications even though this concept is only partially specified in UML, FUML, MOF and SMOF. MOF 2.5 and UML 2.5 state that for ordered association ends, links "carry ordering information in addition to their end values" (see [6, 13.2] and [9, 11.5.3.1]); however, UML does not explicitly define any abstract syntax for links and although such syntax is defined in MOF, ordering isn't. FUML excludes associations with ordered ends (see [5, section 7.2.2.2.22]). SMOF does not define any syntax or semantics for associations with ordered ends (see [**?**, sections 10.1.[56]SMOF 1.0 FTF beta2). Here, support for ordering reflects an implicit assumption in OMG's practice of metamodeling & profiling that at most one association end is ordered.

ModelElement corresponds to the concept of element in FUML[5, section 8.3.2.2.19], MOF[6, sections 9.2, 13.5] and SMOF[7, section 9.1.2.3]. SMOF-like multiple classification is supported with multiple ModelElements for the same uuid, one for each classifying metaclass.

4 Serialization and API

UML 1.x revisions were published with an API specification in terms of OMG's Interface Description Language (IDL)[4] The OMG stopped this practice based on the recommendation from the UML 2.0 finalization task force to "retire 'Model Interchange Using CORBA IDL' as an adopted technology because of lack of vendor and user interest." As part of the 2.0 major revision, the OMG also switched from XML DTDs to XML Schema to support validating models serialized as XML Documents against their metamodel XML Schemas. Several factors contribute to persistent probems of poor model interchange with XMI: the document production rules specified in English and BNF allowed for many serialization options that increased the complexity of XMI implementations to recognize them when loading XMI documents produced from other tools; since the schema production rules have yet to be applied to UML2.x, tool vendors and

[4] For UML 1.5's IDL, see http://www.omg.org/spec/UML/1.5.

user continue to accrue missed opportunity costs due to the inability to validate XMI documents against official XMI schemas. The OMG is keenly aware of these issues. Recent improvements made in Canonical XMI 2.5 minimize but do not eliminate serialization variability and promote but do not ensure serialization reproducibility.

4.1 Normalization yields simpler tabular serialization

The fundamental source of complexity and poor interchange stems from a design decision in the XMI 2.x specification to represent MOF's exclusive ownership principle in terms of nested XML elements: The resulting tree serialization of a model is a materialized view of the model's ownership structure. XML trees are inherently ill-suited for large-scale model management because comparing trees is computationally expensive even with the state-of-the-art Robust Tree Edit Distance (RTED) algorithm whose cubic worst-case runtime complexity is optimal␣[2]. This means that comparing serialized models becomes practically unreasonable for models with millions of elements.

Switching to a serialization paradigm based on the normalized schemas described in Sec. 3 will provide tangible model interchange benefits for end users compared to the current serialization paradigm based on XMI trees due to the improved efficiency of comparing models serialized as normalized tables vs. trees. Comparing trees is computationally expensive: the Robust Tree Edit Distance (RTED) algorithm has a worst-case cubic runtime complexity [2] that is impractical for large models with millions of elements. Comparing normalized tables reduces is computationally reasonable thanks to a worst-case superlinear runtime complexity that should remain practical even for models with billions of elements (rows) [1] with the added benefit that distributed version control systems like GIT should report precise and accurate changes since differences reduce to additions and deletions of table rows.

This switch has a subtle but important implication on the representation of cross references. Since the 2.0 major revision, the OMG XMI specification has distinguished cross-references within a document vs. across documents. In the former case, a cross reference is represented as an XML idref for the XML id of the locally referenced element. In the latter case, the XMI specification allows for five different representations for a cross reference in terms of XML id, XMI uuid, label or potentially arbitrary XLink and XPointer expressions. Most tools follow OMG's serialization practices that rely on XML idref for local and cross references (the XML idref becomes a fragment for an href). Historically, the multitude of options and technologies involved for representing basic element cross references has been a significant source of poor model interchange across tools. Switching to a normalized serialization strategy eliminates altogether the distinction for local vs. cross references and the options for representing them because all elements are referenced, locally or externally, via their uuid. The fact that cross references are represented uniformely regardless of whether they are local or not means that the representation of the normalized serialization is independent of its organization in one or multiple documents. The fact that the normalized serialization produces tabular data without empty columns means that it is possible to take advantage of modern data analytics frameworks for processing model data because the form and content of normalized tabular data is indepedent of how it is organized in terms of documents. This is a significant advantage compared to OMG's XMI tree-based serialization where the form and content of the serialized representation depends on its organization in terms of one or more documents and on their location and vice-versa.

4.2 Separating Modeling and Data APIs

The Eclipse Modeling Framework (EMF) established a model-driven development culture for generating the API of a modeling tool from its abstract syntax metamodel[4], that is, a modeling API is generated from a metamodel. Despite its widespread adoption, the EMF-based modeling API generation paradigm does not address the users needs for interoperability of modeling APIs or interchange of model serializations across different tools. For example, Eclipse UML 5.0.0[5] and MagicDraw UML 18.0[6] implement the same OMG UML 2.5 metamodel[7]. However, their modeling APIs are incompatible, not only because of differences in EMF code generation techniques used but primarily because these generated modeling APIs are tightly coupled with their generated implementation. This means that for a given OMG UML 2.5 metaclass (e.g., Namespace), there is a corresponding modeling API EClass defined in Eclipse UML 5.0[8] and MagicDraw 18.0[9]; however these have nothing in common except for EMF's EObject and Notifier interfaces. In fact, this example illustrates some subtle differences that cause significant API-level interoperability problems when working with EMF-based technologies with these two modeling tools:

- The EMF EModelElement API is important to enable EMF's powerful annotation mechanism␣[4, Sec. 5.6-7]. However, only the generated Eclipse UML metaclasses inherit from EModelElement, the generated MagicDraw UML metaclasses don't. This difference means that many EMF-based techniques that assume that every model element can be annotated will not work as intended when operating on MagicDraw UML models unlike their Eclipse UML counterparts.
- The MagicDraw UML Namespace metaclass shows that it can own MagicDraw UML Diagrams. This is a MagicDraw-specific implementation of the OMG UML 2.5 metamodel that substantially different than the OMG UML 2.5 Diagram Interchange annex[9, B.2.2].
- Comparing class attributes corresponding to association ends in the Eclipse UML and MagicDraw UML APIs can be difficult; it is particularly helpful to have knowledge of the particular EMF code generation techniques involved

[5] See https://wiki.eclipse.org/MDT/UML2/UML2_5.0_Migration_Guide
[6] See http://docs.nomagic.com/display/MD184/UML+2.5+Meta+Model
[7] See http://www.omg.org/spec/UML/20131001/UML.xmi
[8] See http://download.eclipse.org/modeling/mdt/uml2/javadoc/5.0.0/org/eclipse/uml2/uml/Namespace.html
[9] See http://jdocs.nomagic.com/183/com/nomagic/uml2/ext/magicdraw/classes/mdkernel/Namespace.html

to recognize the tool-specific correspondences between CMOF association ends defined in the OMG UML 2.5 metamodel and their corresponding EMF representation in terms of EReferences and/or EOperations in the generated tool-specific APIs.

The root cause of non-existent API-level model interoperability stems from the lack of distinction between two different kinds of APIs: abstract syntax vs. information content schema:

- An abstract syntax API provides support for creating, deleting, updating and navigating across model elements and data according to metaclasses, stereotypes, associations and datatypes defined in metamodels, profiles and libraries. Such APIs are particularly important for model query (e.g. OCL) and transformation (e.g QVT) because they enable expressing queries and transformations in terms of the entities (metaclasses), relationships (associations, properties) and operations defined in the abstract syntax.
- A normalized schema data API provide support for internally representing the information content of a model at an API level independently of its external representation in one of possibly multiple serializations (e.g. XMI, RDF, OWL, Json, CSV, ...). The schemas described in Sec. 3 correspond to a 4th normal form normalization of a database schema [13]. Normalized schemas for the information content of libraries, metamodels, profiles and models enables separating the functional concerns of modeling (via abstract syntax APIs) from the information content concerns of model interchange via serialization to/from external representations such as XML, XMI, RDF, OWL, Json, CSV, etc. Normalization yields tables where each column corresponds to an essential characteristic (a primary key, a foreign key, an attribute); which in turns simplifies serialization matters because there are no optional values and no nulls.

Metamodeling frameworks like EMF provide supprot for generating abstract syntax APIs (and implementations) from metamodels and profiles (e.g., the Eclipse UML tooling). It is likely that OMG's emphasis on XMI Schemas in the major 2.0 revision of MOF, UML and XMI is responsible for demphasizing the importance of specifying the information content of models explicitly as is done in this paper. A significant advantage of the normalized schema APIs described in this paper stems from the possibility of leveraging modern data processing frameworks like Apache Spark for scaling up complex model transformation workflows as described in Fig. 8 taking advantage of the relational form of the normalized schemas for query optimization [14] and of the support for specifying complex model query and transformations in terms of graphs of relational data [15]. In contrast to the affinity of the normalized schemas for optimization and parallelization, a conventional approach with an XMI-based or API-based containment tree representation of models would be technically much more difficult to optimize or parallelize.

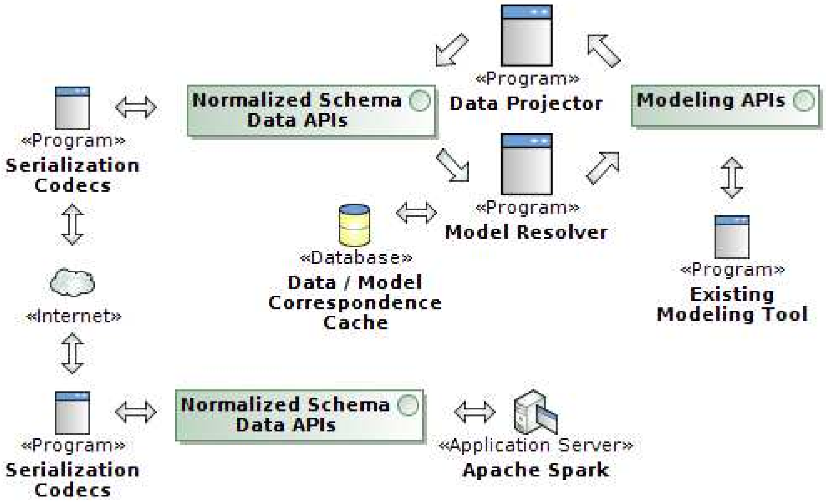

Fig. 8. Proposed architecture to work with existing modeling tools using normalized serialization.

5 Summary

This paper makes three significant contributions towards addressing significant issues with the current paradigm for developing modeling specifications at the OMG. First, this paper carefully explained the intrinsic sources of complexity in OMG's reflexive metamodeling architecture (MOF) where the notion of a profile does not cleanly fit the multi-layered modeling architecture. It is noteworthy that most of the complexity stems from the multiple roles that the concept of property plays in specifying libraries (datatype attributes), metamodels (association ends) and profiles (stereotype association ends). Second, this paper reconstructs a considerably simpler set of schemas for specifying the information content of all modeling artifacts including models that instantiate metamodels with optionally applied profiles. Focusing on the information content instead of the abstract syntax API is key to a significant simplification compared to the current specification practices where a single abstract syntax is used for both generating an API and for tree-based serialization. Third, the paper only sketched a promising area for future work: leveraging powerful data analytics platforms for scaling up complex modeling workflows thanks to the affinity of normalized schemas for optimization and concurrency.

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. Steven S. SKiena. The Algorithm Design Manual, second edition. Springer, 2008.
2. Mateusz Pawlik and Nikolaus Augsten. Efficient computation of the tree edit distance. ACM Trans. Datab. Syst. 40, 1, Article 3 (March 2015).
3. B. C. Pierce, "Types and Programming Languages", 2002.
4. D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, "EMF Eclipse Modeling Framework", 2nd edition, Addison-Wesley, 2008.
5. Object Management Group, "Semantics of a Foundational Subset for Executable UML Models (fUML)", version 1.2.1, formal/2016-01-05, 2016.
6. Object Management Group, "Meta-Object Facility Core Specification", version 2.5, formal/2015-06-05, 2015.
7. Object Management Group, "MOF Support for Semantic Structures", version 1.0 FTF beta 2, ptc/2011-08-22, 2011.
8. Object Management Group, "Report of the UML version 2.4.1 Revision Task Force", ptc/2011-01-19, 2010.
9. Object Management Group, "Unified Modeling Language version 2.5", formal/2015-03-01, 2015.
10. Guarino, Nicola and Chris Welty. 2004. An Overview of OntoClean. In Steffen Staab and Rudi Studer, eds., The Handbook on Ontologies. Pp. 151-172. Berlin:Springer-Verlag
11. Jiří Procházka, Richard Cyganiak, Toby Inkster, Bob Ferris. The Property Reification Vocabulary 0.11, http://smiy.sourceforge.net/prv/spec/propertyreification.html
12. Paul V. Biron, Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004. http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/
13. William Kent. A Simple Guide to Five Normal Forms in Relational Database Theory. Communications of the ACM 26(2), p. 120-125, Feb 1983.
14. Michael Armbrust *et al.* Spark SQL: Relational Data Processing in Spark. SIGMOD'15, 2015.

15. Joseph E. Gonzalez *et al.* GraphX: Graph Processing in a Distributed Dataflow Framework. OSDI;14, 2014.