Viewing the OML as a Variant of the UML

Brian Henderson-Sellers[1], Colin Atkinson[2], and Don Firesmith[3]

[1] University of Technology, Sydney, PO Box 123
Broadway, NSW 2007, Australia
brian@socs.uts.edu.au
[2] Fraunhofer Institute, Kaiserslautern, Germany
atkinson@iese.fhg.de
[3] Lante Corporation, Dallas, USA
FiresmithD@aol.com

Abstract. The OPEN Modelling Language, OML, was published during the standardization process which finally led to UML version 1.3. While being contributory to this process, there are still some features of the OML which have not been adopted in the current version of the UML. These features offer capabilities which are complementary to those of the UML. This paper describes how these features of the OML can be made available to UML developers by viewing the OML as a variant of the UML.

1 Introduction

The UML [1] and OML [2] are two object-oriented modeling languages which were both developed in response to the unease in the software industry about the growing divergence of object-oriented methods, and the often unnecessary differences in object-oriented modeling notations. Both notations represent an attempt to capture the core concepts of object-orientation and standardize upon a set of intuitive graphical icons. As such, they share many core concepts and, in fact, have many common roots. However, there are certain areas in which the OML and UML do differ significantly, and where UML developers may benefit directly from the OML features not currently supported in the OMG standard. Alternatively, OML developers may benefit from access to UML features not supported in OML.

Given the large overlap between the core concepts, it would seem desirable to provide object-oriented modelers with the union of features in the OML and UML. Fortunately, the UML provides a couple of ways to extend the features of the UML with new concepts: one is called a UML variant and the other a UML extension.

An extension uses special "built in" features at the M1 level (Fig. 1). These features are stereotypes, tagged values and constraints, together with appropriate notational elements. The changes are made at the model level. The UML documentation contains two such pre-defined extensions: one for business engineering and one for supporting the Objectory process [1,3]. So for instance, we

Robert France and Bernhard Rumpe (Eds.): ≪UML≫'99, LNCS 1723, pp. 49–66, 1999.

might choose (as does Objectory: [1] (p4-7)) to specialize the class concept into boundary classes, entity classes and control classes by application of an appropriate stereotype at the M1 (model) level. The resulting combination of UML and these additional user-defined stereotypes is known as a "UML extension".

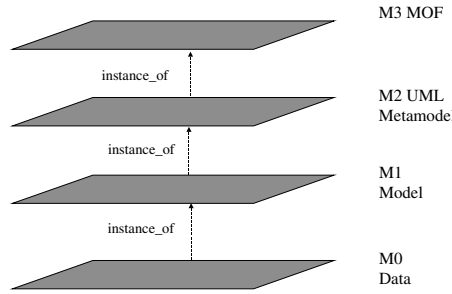

Fig. 1. UML's four layer architecture

A UML variant, on the other hand, extends the UML metamodel at the M2[4] level i.e. the metamodel itself. The variant uses the existing architecture of the metamodel for UML and adds concepts (metatypes) to the metamodel. The resulting metamodel is known as a "UML variant".

In a paper of this size it is not possible to give a complete specification of an OML extension/variant to the UML. The focus of this paper is rather to describe the merits of doing so, discuss the relative pros and cons of using the variant or extension mechanism, and to illustrate what form such a variant would take. Following a brief background-setting overview of the OML (Sect. 2), we then give, in Sect. 3, a technical description of some of the metalevel features of UML and OML, and the general nature of the difference between them. We also describe a number of features of the OML which we believe would be particularly beneficial to UML developers. In the following two sections (Sects. 4 and 5), we describe why the UML variant approach seems to make more sense than the UML extension approach for this purpose and also describe, in Sect. 5, how such a variant would be defined. In Sect. 6, we extend the discussion to propose the use of conformant and non-conformant variants.

2 The History of OML and Future Contributions to the UML

OML [2], was published in early 1997 during the standardization process which finally led to the (current) UML Version 1.3 [1]. While contributing to that process, there are still some features of OML which have not yet been adopted

into UML. An overview of the comparative features of OML, as compared to UML, was given in [2,5] and in more detail in [6].

OML has been influenced by pure OO approaches such as RDD [7] and Eiffel [8] but at the same time remains a completely programming language-independent modeling language. OML Version 1.0 (published in 1997) was amended slightly in 1998 [9,10] to bring it into alignment with the, by then OMG-endorsed, UML Version 1.0. Despite the large overlap between the OML and the UML, there are still some useful features of the OML which are not currently adequately supported in the UML and which UML users may find helpful for their modeling work. In this paper, we identify these features and explain how the most important OML-specific features may be reconciled with the OMG standards in the form of a UML variant. By presenting these features as an extension to the UML, we can make these OML features available to developers using UML in their development project. The long-term goal would be to offer these modifications to the OMG for potential inclusion in future versions (e.g. Version 2.0) of the OMG/UML standard.

3 Key OML and UML Metamodel Fragments

3.1 UML

UML is characterized by, and emphasizes, use cases, relationships as reifiable classes, an obvious data modeling heritage, the use of bidirectional associations and rôles on association ends from OMT and an increasing reliance on stereotypes.

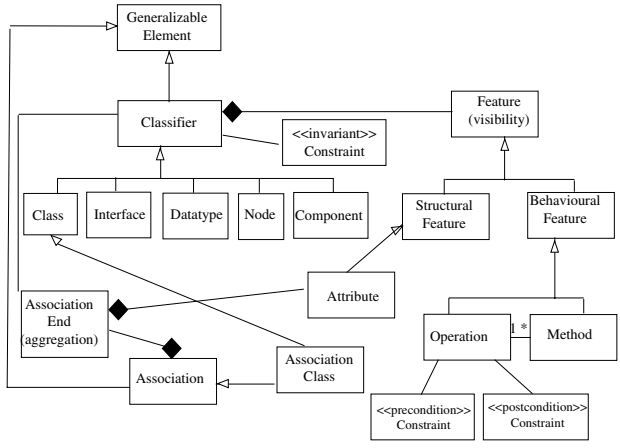

Fig. 2. Main structural elements of the UML metamodel (Version 1.3)

Fig. 2 shows the static architectural model for UML in which there are five major metalevel concepts, collectively called Classifier: Class, Interface,

Datatype, Node and Component. In UML, an *object* is "an instance that originates from a class" [1] (p2-90), but is not defined in the core model package. An object in UML has only attributes and no operations [1] (p3-53).

Interface is shown as a subtype (using the Generalization relationship) of Classifier which means, according to the definition of Generalization in [1] (p2-34) (see also below), that the Interface has all the characteristics of its supertype, Classifier. Since a Classifier has attributes, methods and operations while an Interface has only operations, this would appear to be inconsistent with the axiomatic definitions, meaning that the UML metamodel is not applied correctly in describing itself.

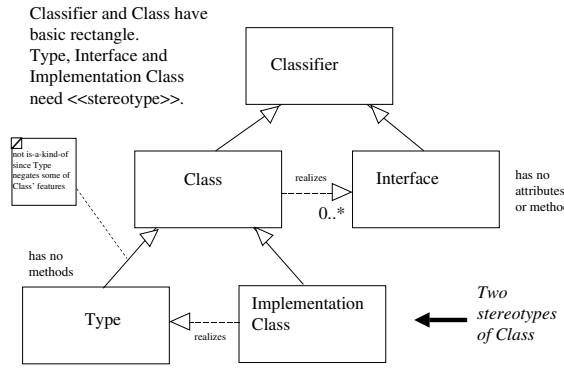

Fig. 3. Relationships between the UML metaclasses Classifier, Class, Interface and the two Class stereotypes: Type and Implementation Class

While Class and Classifier are full metalevel concepts (metaclasses), there are other relevant stereotypes: ≪type≫ and ≪implementationClass≫ (Fig. 3) where the ImplementationClass is said to realize the Type. One area of current concern with this model is that, once again, the nature of a stereotype is that of specialization inheritance (a.k.a. Generalization), since stereotyped instances have "the same structure (attributes, associations, operations) as a similar non-stereotyped instance of the same kind" [1] (p2-66). Thus, in Fig. 3, we should be able to say that "a type is a special kind of class". This is, however, not true since a Class can have Methods but a Type cannot. Thus the Type metaclass (shown as such in Fig. 3) subtracts from its "parent", the Class metaclass (as does Interface from Classifier) — this subtraction technique with "inheritance" was recognized many years ago by Brachman [12] as a bad modeling strategy.

Classifiers are composed of (black diamond notation) features which have an associated visibility (Fig. 2). Features may be structural or behavioral. Behavioral features are either operations or methods, where methods implement

operations. Structural features are only attributes. Assertions can be included by adding stereotyped Constraints to operations and classifiers as shown.

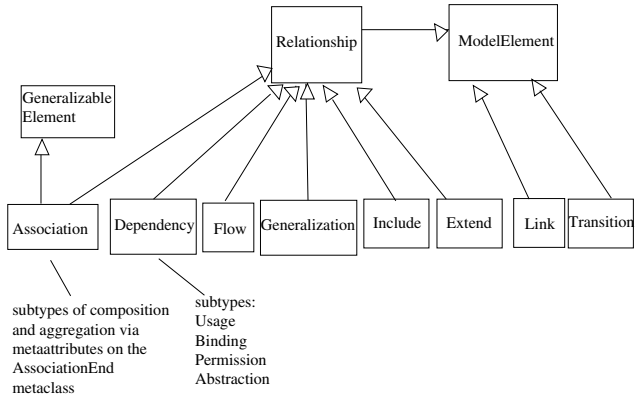

Fig. 4. Metamodel fragment for relationships in UML Version 1.3

Fig. 4 shows the Version 1.3 metamodel for UML relationships. Relationship is an abstract metaclass with no specific semantics which defines "a connection among model elements" [1] (p2-41). Subtypes of Relationship are Association, Dependency, Flow and Generalization. In addition, Include and Extend are metaclasses in the Use Cases package which also inherit from Relationship (in the Core package). Extend and Include are also said to be directed relationships although this does not seem to be enforced anywhere in the metamodel.

Of the metasubtypes, an Association is said to be "a semantic relationship between classifiers" [1] (p2-19) which defines a set of tuples relating the instances of these classifiers. It is bidirectional in nature. A Dependency is a "term of convenience for a relationship other than an Association, Generalization, Flow, or metarelationship" [1] (p2-30). It is a undirectional (or directed) relationship — but the unidirectionality is not enforced. There are four kinds of Dependency: Abstraction, Binding, Permission and Usage (shown with stereotype labels in the notation). Abstraction may be unidirectional or bidirectional and has four predefined stereotypes: Derivation, Realization, Refinement and Trace (also shown with a single stereotype label in the notation). While ≪realize≫ may be unidirectional or bidirectional, ≪derive≫ is unidirectional whereas for ≪trace≫ it is said that the "directionality of the dependency can often by ignored" [1] (p2-19). Binding, Permission and Usage are unidirectional and the last two have several pre-defined stereotypes each. Flow represents a relationship between two versions of an object and is a directed relationship. It has two pre-defined stereotypes.

The Generalization relationship is a "taxonomic relationship between a more general element and a more specific element. The more specific element is fully

consistent with the more general element (it has all its properties, members and relationships) and may contain additional information." It is "a subtyping relationship (i.e. an Instance of the more general GeneralizableElement may be substituted by an Instance of the more specific GeneralizableElement)" [1] (p2-34). It has one stereotype, ≪implementation≫ which is all Generalization is *minus* the support for the interface and for substitutability (see Fig. 5(b)). It therefore violates its own metasupertype's axiomatic definition!

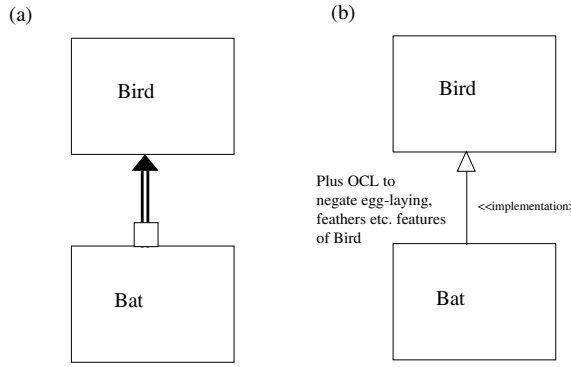

Fig. 5. Modelling Bat as a subclass of Bird because both can fly: (a) using OML, implementation (or white box) inheritance is used directly where this relationship is a subtype of Inheritance and a peer of Generalization; and (b) using UML, implementation inheritance is a stereotype of Generalization which means that, *de facto*, it has the same properties as its superclass, Generalization. Unwanted features of Generalization have then to be negated away by use of appropriate OCL constraints.

Link and Transition are also shown in Fig. 4. While these are not part of the Core package's Relationship hierarchy, they are intimately connected (as are Include and Extend from the Use cases package). A Link is simply an instance of an Association. It is a subtype of ModelElement in UML Version 1.3, not of Relationship. The Transition metatype is also a subtype of ModelElement but in this case it would appear from the UML documents [1] (p2-132) that it should in fact be a subtype of Relationship since it is defined to be "a directed relationship between a source state vertex and a target state vertex" in the state machine metamodel.

Rôles in UML have two meanings: (i) as a label on the AssociationEnd or (ii) as AssocationRole, AssociationEndRole or ClassifierRole in the collaboration diagram. As a label on the AssociationEnd it is "a name string near the end of

the path" [1] (p3-61); a concept which does not seem to be supported in the metamodel itself. In the collaboration diagram, the classifier rôle describes how a specific participant (interface) in a collaboration may play a specific rôle. It is effectively a viewpoint on an object in the specific context of the collaboration in question. A ClassifierRole thus defines a set of Features which are themselves a subset of those in the base Classifiers. AssociationRoles and AssociationEndRoles are the corresponding usages of Associations/AssociationEnds in the context of a collaboration. The two different meanings of rôle are described in [13] (p414) as the static and dynamic aspects of rôles.

In summary, the fragments of the UML metamodel[1], described above, to which OML offers extension or modification, relate to the focal points of (a) class/type/interface metamodel structure, (b) responsibilities, (c) relationship hierarchy — especially aggregation relationship, inheritance stereotypes and dependencies and (d) rôle modeling.

3.2 OML

OML is characterized by a balanced use of use cases, responsibilities and rôle modeling. In OML, all relationships are unidirectional to preserve encapsulation/information hiding [15]. Stereotypes are used similarly to UML.

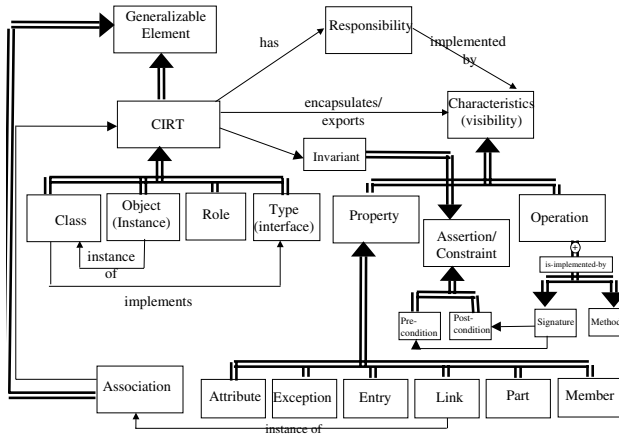

Fig. 6. OML Version 1.1 metamodel — incomplete fragment of the static architecture with the same scope as Fig. 2

Fig. 6 shows the core elements of the static metamodel, laid out in the same way as Fig. 2 for ease of comparison. Ignoring terminological differences, we note strong similarity except that the CIRT supertype is concrete (with its own

[1] We restrict our discussion here to aspects of the static, architectural components of both the UML and OML metamodels.

notation) and its subtypes are not identical to those of the UML Classifier. CIRT stands for Class or Instance or Rôle or Type. While Class and Type map roughly to UML's Class and Interface, the three UML subtypes of Datatype, Node and Component are missing (but could easily be added or retained as stereotypes) and OML has two subtypes NOT in the UML metamodel: Instance (Object) and Rôle. In strict metamodeling[2], Instance and Type/Class would not appear in the same Mx layer (Fig. 1). On the other hand, the appearance of Rôle as a subtype of CIRT is purposeful since there is a need to support rôle modeling in the class diagram as well as in the collaboration diagram. Also of note in Fig. 6 is the fact that OML eschews the ideas of AssociationClass and AssociationEnd as distinct metaclasses.

The structure of the Feature/Characteristic hierarchy is also richer in OML. Property has more than the single subtype of Attribute as in UML (Fig. 2). These other subtypes relate to the more extensive use of association and aggregation modeling in OML (see below).

A very important metaclass in OML is that of Responsibility (Fig. 6). Classifiers/CIRTs have high level responsibilities each of which is linked to/realized by one or more features (which may be structural or behavioural). Responsibilities are adopted from the work of [7] and carry significant semantics — unlike the responsibility notion in UML which is a stereotyped comment in Version 1.3 (tagged value applied to Classifier in Version 1.1).

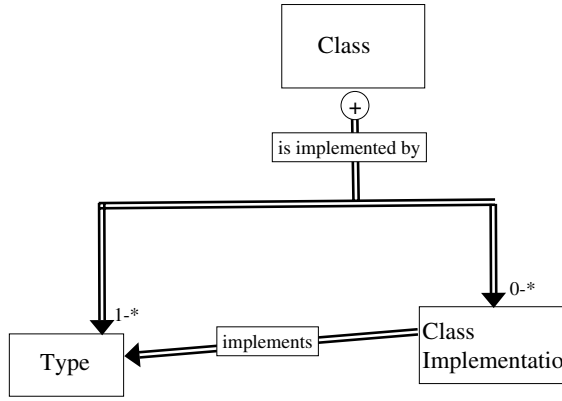

Fig. 7. In OML, the definitional relationship between Class, Type and Class Implementation is that of aggregation not inheritance

In OML, Types and Interfaces are not strongly differentiated in the metamodel. A Type is defined [2] (p17) to be a declaration of visible characteristics

[2] Neither UML nor OML employ strict metamodeling.

(= UML Features) that form all or part of the interface. This set of characteristics defining the type is therefore a subset of those defining the interface. In OML, it is not only objects and classes that can have interfaces but also use cases, packages etc. Thus if a class has a single type, type is identical to interface. Since one (type) is a subset of the other (interface), only one concept (metaclass) is retained in the OML metamodel. This is in contrast to UML where a set of operations define a service. This set is called the Interface, which is roughly analogous to Type in OML.

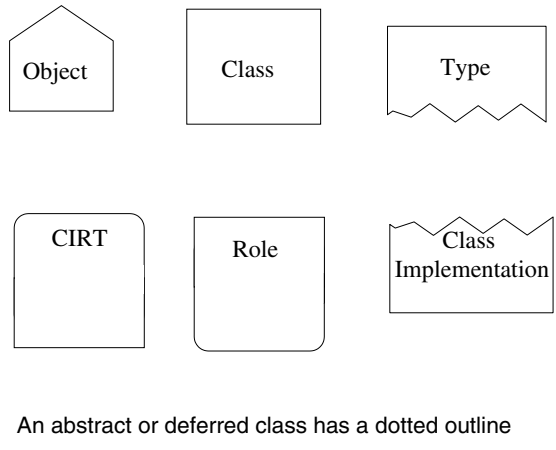

Fig. 8. Stereotypes in UML may be given their own graphical icons. Here are some of those suggested in OML

Secondly, since Type in OML is the declaration of the external view or specification, and since the full Interface consists of one or more Types[3], then Interface can be considered as redundant and the totality of the Class is in fact a combination (or aggregation) of this Type and its Class Implementation (Fig. 7). Thus, instead of generalization, OML uses aggregation to link together the concepts of Type (inclusive of Interface), Class and Class Implementation (Fig. 7 compared to Fig. 3). In the OML notation, the various stereotypes of Fig. 3 are all given icons, as permitted within the OMG standard (Fig. 8). These were chosen based on semiotic[4] principles to make learning easier and more intuitive.

The original relationship metamodel for OML made a clear distinction between four sets of relationships: referential and definitional (as used in class diagrams/semantic nets), transitional (state models) and scenario (use cases di-

[3] The use of the names Type and Interface in UML and Java is opposite to that in OML

[4] The study of signs and symbols

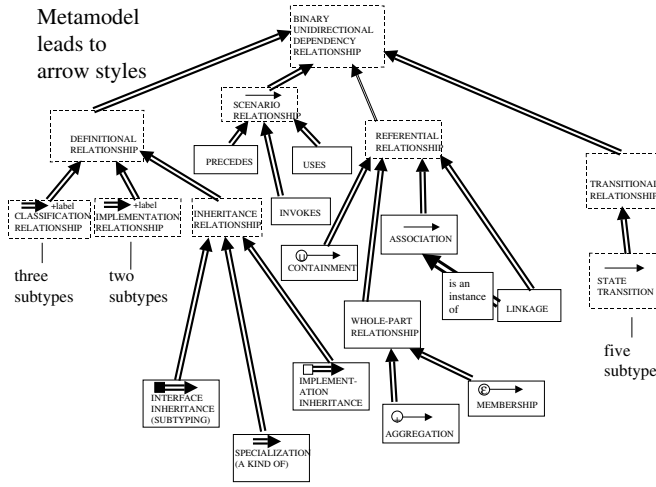

Fig. 9. Fragment of the OML metamodel hierarchy showing all relationships — updated from [2] to Version 1.1 based on [9]

agrams etc.). In turn, these four metaclasses have a number of subtypes (Fig. 9). Of specific interest here are

- all relationships are binary and *unidirectional*
- all relationships are dependency relationships (tying in with their unidirectionality)
- aggregation (Fig. 10), membership and containment are clearly defined subtypes of association
- there are three types of inheritance relationship: generalization/specialization (a-kind-of), interface (blackbox or subtyping) and implementation (whitebox) — all peers.

Since all relationships are unidirectional, they are arrowed to indicate the direction of dependency. Thus associations have a single direction which means that an unarrowed association can be given the meaning of "TBD" (to be decided) — a useful modeling tool when doing rapid analysis and design sketches of the emerging model. Additionally, bidirectional relationships are only a shorthand for a pair of unidirectional relationships that are semi-strong inverses of each other. The iconic representation of the three specific subtypes of Association (Membership, Containment and Aggregation) offers visual differentiation. The black and white box on two of the subtypes of inheritance is another valuable visual reminder (Fig. 5(a)).

OML's specialization inheritance (Fig. 9) is in full agreement with the UML definition of Generalization (see Sect. 3.1). However, generalization is only one kind of "inheritance", the others being interface inheritance and implementation

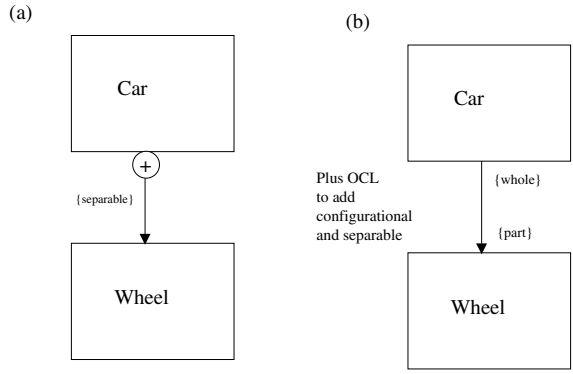

Fig. 10. Modelling the commonest type of whole–part relationship: (a) directly in OML using the configurational symbol (plus in a circle) and the {separable} stereotype; (b) in UML — because neither black nor white diamond can be used to describe a configurational/separable whole–part relationship, it has to be constructed from a regular association to which is added: (i) a {whole} and a {part} constraint, (ii) a navigability arrow and (iii) some OCL constraints to make the association both configurational and (iv) separable.

inheritance (Fig. 5(a)) — although in practice the first two are often purposefully confounded. These three (or pragmatically two) form a partition of an (abstract) superclass in the metamodel (called Inheritance Relationship in Fig.9).

4 The UML Extension Mechanism

The three extension mechanisms available in UML are stereotypes, tagged values and constraints. Some of OML's characteristics could possibly be re-expressed with stereotypes. In particular, Table 1 shows the necessary stereotypes together with the metamodel class which they extend in OML.

Although defined at the model or M1 level, a stereotyped class can thus be thought of as a "virtual" or "pseudo" M2 class which partitions an existing M2 metaclass. In other words, we might describe a stereotype as creating an implicit user-defined metasubtype.

Thus, OML requires a stereotype ≪rôle≫ for Classifiers to permit their use in Class diagrams as well as the existing support in Collaboration diagrams — although a new M2 metasubtype would be much more powerful.

OML has three distinctive kinds of inheritance which could be given stereotypes. The problem here is that specialization, specification and implementation

Table 1

Metamodel Class	Stereotype (submetaclass)
Class	Rôle
Generalization	White box inheritance
	Black box inheritance
	Specialization
Association	Whole–Part (WP) Relationship
WP Relationship	Configurational (a.k.a. aggregation)
	Membership
Association	Containment

inheritance really create a single partitioning rule i.e. they should be peers, together with an abstract supertype. To create this in UML would require a variant not an extension (see Sect. 5).

OML has strong support for "aggregation" (configurational whole–part (WP) association relationship) and "membership" (non-configurational WP relationship). One possible way to represent these using stereotypes would be to add a ≪WPRelationship≫ stereotype to Association (with aggregationKind set equal to none) from which two additional stereotypes of ≪configurational≫ and ≪membership≫ could be created. The third new Association type, Containment, is then a stereotyped Association. On the other hand, a more semantically powerful representation in the metamodel is discussed in Sect. 5 using the idea of a UML variant.

In addition to showing the stereotypes of Table 1 as keywords in guillemets, we recommend the following new icons: rôle: Greek tragedy mask (Fig. 8); kinds of inheritance: white and black box options (Fig. 9); WP relationship: annotations at client end of relationship (Fig. 9); and Containment: annotation at client end of relationship (Fig. 9).

In conclusion, whilst some of OML's constructs can be readily represented as stereotypes, many of these can more cleanly use the variant ideas supported in UML.

5 OML as a UML Variant

In this section, we describe the OML model elements which cannot be simply created at the model level (M1) by judicious use of user-defined stereotypes. Instead, the metamodel (M2) requires modification, thus creating a UML variant.

5.1 Responsibilities

In OML (Fig. 6), a CIRT has Responsibilities which are implemented by Characteristics. In the variant version of UML, we introduce a new metaclass called Responsibility which has a meta-association to Classifier and a meta-association to Feature (Fig. 11 which shows the relevant fragment of Fig. 2, updated in this

way.) This replaces the current UML responsibility which is (a) a stereotyped comment with no semantics and (b) confused with the notion of a contract e.g. [11,16].

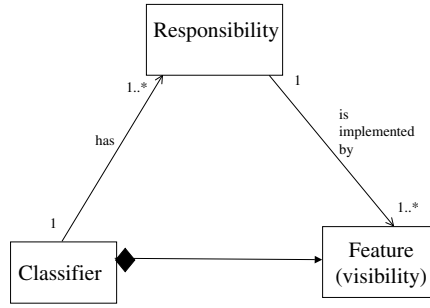

Fig. 11. Addition of Responsibility metatype to UML variant

Notation for responsibilities is already available in UML (adopted from OML: [17]). Responsibilities are documented in a fourth box on the class icon. What is required (see below) are well-formedness rules to ensure that the Class–Responsibility–Operation links are correct.

5.2 Aggregation

Although we have shown in Sect. 4 how whole–part relationships can, to some degree, be represented by the use of stereotypes in a UML extension, a cleaner model can be derived by judicious modifications to the metamodel itself. The following changes would be needed:

- in the AssociationEnd metaclass, the aggregation: aggregationKind meta-attribute should be erased.
- the introduction of a new metaclass called Whole–Part (WP) Relationship. While this can be regarded as a kind of Association, it is important that it inherits from a unidirectional relationship rather than from the bidirectional Association metaclass. This suggests that, despite the clear is-a-kind-of connection between it and the Association metaclass, the new WP Relationship metaclass might best inherit from either Dependency or Relationship.
- the addition of two new subtypes of Whole–Part Relationship metaclass: (a) Configurational and (b) Membership
- the introduction of a new subtype of Association called Containment

Annotation for whole–part and containment relationships are given already in OML. These could be used “as is” (Fig. 9). The Whole–Part Relationship metaclass also needs additional well-formedness rules. These would formally express the mandatory existence of (i) emergent property, (ii) resultant property,

(iii) irreflexivity and (iv) asymmetry ([18]). In addition, careful formal definition of containment, which is *not* a whole–part relationship, is needed — again we encourage the evaluation and derivation of appropriate well-formedness rules.

5.3 Type/Class/Interface

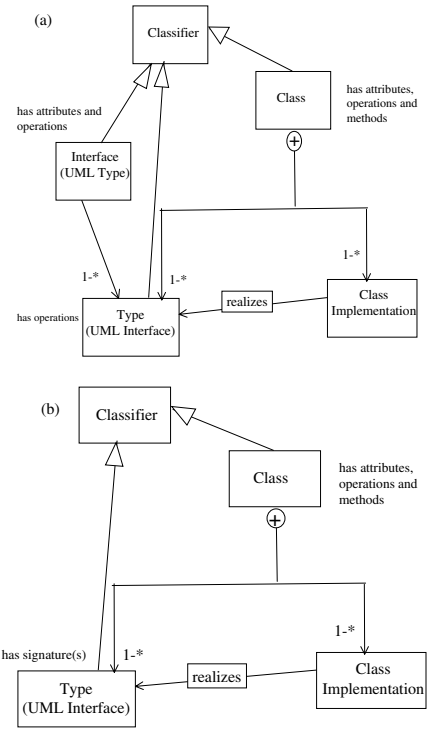

Fig. 12. Aggregation relationships between Class, Type and Class Implementation linked into the UML architecture involving Interface and Classifier metaclasses

The tidiest way to improve the metamodel of Fig. 3 would be a full revision using correct Generalization relationships. In making such a drastic change it might be better to totally revise this fragment of the metamodel. One suggestion is given in Fig. 12(a). It can be seen that a Class is made up from the specification (or Interface) which consists of several Types together with the Class Implementation. If required, an (OML) Interface is then equal to one-to-many Types[5]. However, a better model (Fig. 12(b)) might be one in which Type and

[5] This means that an alternative, but equivalent, model could be drawn in which a Class is made up of a single Interface plus one-to-many Class Implementations

Interface are fused together. This metaclass then has one or more signatures (no operations and no attributes).

If an inheritance hierarchy is preferred, then initially it seems possible to make an Interface (UML Type) to inherit from (Generalization relationship) Type (UML Interface) since a UML Interface has operations while a UML Type has not only operations (and their corresponding signatures) but also attributes. However, since Fig. 12 shows that an Interface (UML Type) is equivalent to one-to-many Types (UML Interfaces), a Generalization relationship is clearly inappropriate. Yet the need for a Classifier metaclass remains. However, for it to be a supertype of Interface, Type and Class (as is probably deemed preferable), the definition of Classifier needs to be modified. At present, a Classifier, like a Class, contains attributes, operations and methods. We propose that these elements are deferred to the Class, such that a Classifier is more of a place holder as an abstract class in the hierarchy, representing just that: model elements that represent the abstraction technique known as classification. If the hierarchy is constructed in the way suggested by Fig. 12, then all inheritance relationships are truly generalization and the purity of the metamodel (i.e. being defined using its own rules) is obtained.

It should be noted, however, that in a sophisticated metamodel such as the UML metamodel, there are likely to be other complications resulting from such a change. Nevertheless, so far as the OML variant of UML is concerned, this slight modification to Fig. 3 *does* result in an acceptable and usable definition of Class, Type and Interface — although it should also be noted that the names of Type and Interface in Fig. 12 are UML nomenclature and there is still the terminological argument between "interface" and "type" in the more general OO modeling community.

5.4 All Relationships Are Dependencies

While the original (Version 1.0, 1.1) relationship metaclasses were distinct, in UML Version 1.3 a partial unification has taken place as shown in Fig. 4. As well as requiring Transition to inherit from Relationship rather than ModelElement[6] in OML, some discussion of Association and Dependency is needed. In UML, Association is bidirectional and the others (apart from some subtypes) are unidirectional. In OML, all relationships are binary (whereas ternary are permitted in UML's associations), undirectional and dependency relationships. Thus, rather than use Association as the base class, the OML variant will focus on Dependency (which is already unidirectional). From this will be constructed the model elements of Fig. 9. The major elements of OML are shown in Fig. 13. The UML Dependency and Relationship metaclasses are fused together into the root Dependency Relationship metaclass (abstract) and the newly introduced Referential Relationship is an abstract metaclass acting as a place-holder. To avoid

[6] We presume this is an error in the Version 1.3 draft documents since the text suggests that in UML V1.3 it was always intended that Transition should inherit from Relationship not from ModelElement as shown in the metamodel.

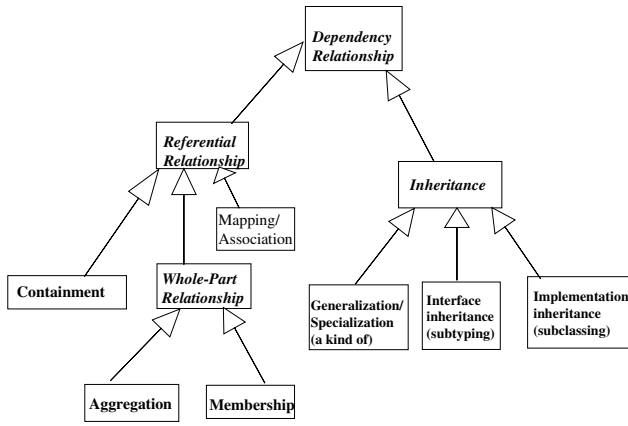

Fig. 13. Suggested OML variant structure for relationships which take the UML Relationship hierarchy of Fig. 4 but refocus on Dependency, ignoring Association and AssociationEnd, and introducing a new Whole-Part Relationship together with its subtypes. Similarly, three subtypes of inheritance are used such that the Generalization metaclass becomes an abstract class, renamed Inheritance for clarity

name clashes in the namespace, OML Association has been renamed Mapping. Association and AssociationEnd are thus not part of the OML variant of UML.

6 Conformant and Non-conformant Variants

The basic premise of a UML variant is that additions should be made at the M2 level. It is implicit that these are *additions* as opposed to changes. Some of the suggestions in Sect. 5 fall into this category. However, other suggestions require changes rather than additions at the M2 level. We may wish to discriminate between these two uses of the word variant by qualifying them. The first we will call a *conformant variant* and the second a *non-conformant variant*. The ideas to improve the relationship and type/class architectures fall into this second class of non-variance. On the other hand, it is feasible to use the existing UML constraint mechanism to eliminate or modify existing metaclasses — in which case the conformant/non-conformant discrimination vanishes. Nevertheless, we will, for the present, label OML as a non-conformant variant of UML — although it contains elements which are extensions and elements which are conformant variants as well.

The possibility of adding the idea of a non-conformant variant to the UML permits evolution in a more flexible fashion than the two current extension mechanisms which both insist that every fragment of the current UML is "correct"

and inviolate. The difficulty in adhering to this strict requirement is evidenced by the significant non-conformant changes made to the original UML metamodel by the Revisionary Task Force (RTF) — for instance the welcome addition of the Relationship metaclass in Version 1.3 (not seen in Version 1.1) and the total revision of the use case association types in Version 1.3 (now stereotyped Dependencies rather than the stereotyped Generalizations of Version 1.1).

A major area of concern remains in the use of Generalization relationships in the diagrammatic, and therefore semantic, definition of the UML metamodel. The definition is good but the use of it is clearly seen to be incorrect. It is vital that (non-conformant) checks of *all* uses of Generalization in the UML Version 1.3 metamodel be made (in preference to "patches" involving overwriting constraints) and, once a consistent definition of black and white diamond aggregation has been accepted (see, e.g., suggestions of [20]), the uses of aggregation and composition also need to be carefully examined in the metamodel definition.

Thus the introduction of the idea that suggested changes to the UML metamodel may in fact be non-conformant variants (i.e. improvements to the metamodel which extend and at the same time change/correct it) could be very valuable in both tightening up the UML and also creating a path forward for its future evolution.

7 Summary and Conclusions

A comparison of specific fragments of the published UML and OML metamodels has permitted us to identify the new stereotypes and metamodel changes necessary to permit the OML to be viewed as a UML variant. Finally, we discussed the potential for the introduction of both conformant and non-conformant variants. Non-conformant variants open up the opportunity for true evolution of the UML. OML is one possible step in that direction.

References

1. OMG: OMG Unified Modeling Language Specification (draft), Version 1.3 alphaR2, January 1999 (unpubl.) (1999)
2. Firesmith, D., Henderson-Sellers, B., Graham, I.: *OPEN Modeling Language (OML) Reference Manual*, SIGS Books, New York, 276pp (1997); Cambridge University Press, New York (1998)
3. OMG: UML Extension for Objectory Process for Software Engineering. Version 1.1, 1 September 1997. OMG document ad97-08-06 (1997)
4. Atkinson, C.: Supporting and applying the UML conceptual framework. Procs. ≪UML≫'98 (1998) 1–11
5. Henderson-Sellers, B.: OML: proposals to enhance UML. Procs. ≪UML≫'98 (1998) 319–329
6. Henderson-Sellers, B., Firesmith, D.G.: Comparing OPEN and UML: the two third generation OO development approaches. Inf. Software Technol. **41** (1999) 139–156
7. Wirfs-Brock, R., Wilkerson, B., Wiener, L.: *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 368pp (1990)

8. Meyer, B.: *Eiffel: The Language*, Prentice Hall, New York, 594pp (1992)
9. Firesmith, D.G., Henderson-Sellers, B.: Upgrading OML to Version 1.1: Part 1. Referential relationships. JOOP/ROAD **11(3)** (1998) 48–57
10. Henderson-Sellers, B., Firesmith, D.G.: Upgrading OML to Version 1.1: Part 2 — Additional concepts and notations. JOOP/ROAD **11(5)** (1998) 61–67
11. Wirfs-Brock, R.J.: Adding to your conceptual toolkit: what's important about responsibility-driven design. Report on Object Analysis and Design **1(2)** (1994) 39–41
12. Brachman, R.J.: "I lied about the trees" or, defaults and definitions in knowledge representation. The AI Magazine **6(3)** (1985) 80–93
13. Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, MA, 550pp (1999)
14. OMG: UML Notation. Version 1.1, 15 September 1997. OMG document ad/97-08-05 (unpubl.) (1997)
15. Graham, I.M., Bischof, J., Henderson-Sellers, B.: Associations considered a bad thing. J. Obj.-Oriented Programming **9(9)** (1997) 41–48
16. Meyer, B.: Applying "design by contract". IEEE Computer **25(10)** (1992) 40–51
17. Booch, E.G.: public communication, Sydney, 19 April 1999
18. Henderson-Sellers, B., Barbier, F.: What is this thing called aggregation?. TOOLS29 (eds. R. Mitchell, A.C. Wills, J. Bosch and B. Meyer), IEEE Computer Society Press (1999) 216–230
19. Booch, G., Rumbaugh, J., Jacobson, I.: *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, USA, 482pp (1999)
20. Henderson-Sellers, B., Barbier, F.: Black and white diamonds. *Procs. ≪UML≫'99*, Fort Collins, CO, October 1999 (1999), this volume