Some problems with the UML V1.3 metamodel

Brian Henderson-Sellers
University of Technology, Sydney
Broadway, NSW, Australia
brian@it.uts.edu.au

Abstract

There are several well-known problem areas in the UML (Version 1.3). Although many of these have been reported to the OMG's Revisionary Task Force (RTF), that venue is highly pragmatic; here we consider the detailed "theory" relating to these problem areas. We first discuss the widely used concept of stereotypes and then concentrate on the metamodel for (i) WP relationships, (ii) Responsibilities; and (iii) Type/Interface/Class. In addition, we identify areas of the metamodel where the UML has been used inexpertly, particularly in the incorporation of generalization to incorrectly represent implementation inheritance relationships and the use of black diamond composition for relationships that are poorly argued to be whole-part relationships. Other issues discussed here include the default directionality of associations; the relationship of an association to a dependency; and the general observation that the UML uses a notation that supports expressibility but is not in itself expressive.

1. Introduction

The development of the Unified Modeling Language (UML) has occurred over the last few years, recently under the aegis of the Object Management Group and specifically the Revisionary Task Force (RTF), now responsible for the UML's evolution. The current version is 1.3 (since June 1999: [27]) but there are plans to issue Version 1.4 in late 2000 and Version 2.0 is being discussed. The deadline for the RFI (Request for Information) for Version 2.0 was late 1999. This request was to identify problem areas rather than solutions to those problems. In this paper, we focus on a selection of issues raised by one particular response [28], discuss the theoretical basis for some of the concerns expressed in that response and go on to propose some possible solutions. Some of these ideas originate in the OPEN Modelling Language (OML: [10]) and have been described largely *in the context of OML* [15]. Here, we further this "technology transfer" to see how these ideas might be moved from the existing OML suite of concepts as a "UML variant" [20] into the UML V1.3 as it migrates towards Versions 1.4 and 2.0.

The areas focussed upon in this paper include (Section 2) the comparative utility of subtypes and stereotypes; (Section 3) whole–part (WP) relationships; (Section 4) responsibilities; and (Section 5) type/interface/class. In addition, in Section 6 we identify areas of the metamodel where the UML has been incorrectly used, particularly in the incorporation of generalization to represent implementation inheritance relationships and the use of black diamond composition for relationships that are poorly argued to be whole-part relationships. The paper concludes with an outline of a few other miscellaneous areas of concern (Section 7).

2. Subtypes and Stereotypes

The UML has three extension mechanisms (stereotypes, tagged values and constraints). Of these, stereotypes are the least understood and potentially confusing and are likely to provoke controversial discussion [2]. In order to understand stereotypes, the contextual background of the 4 layer architecture of the UML first needs to be briefly outlined.

The UML is based around four layers[1] of modelling/metamodelling. The levels are denoted as

- M0 — the object or instance level
- M1 — the (normal) class level a.k.a. model

[1] Atkinson and Kühne [2] note a problem in UML is that the notation, in particular, focusses on the M0/M1 levels such that, when discussing M2 metamodelling ideas, the notation cannot be easily extended. For example, UML's use of underlining to mean instance-of is clearly M0/M1 [1] whereas, in the use of stereotypes, we need to discuss instance-of between layers M1 and M2 rather than M0 and M1.

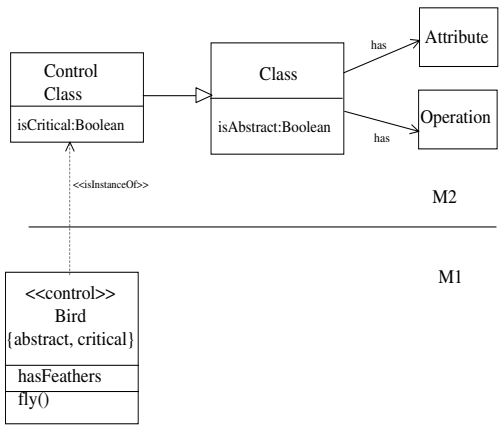

Figure 1. Example of a "virtual" submetatype (ControlClass) and its relation to the predefined metaclass Class and an instance at the M1 level i.e. a normal Class (Bird)

- M2 — the metamodel level at which the UML is defined in detail
- M3 — the metametamodel level which aligns with the OMG's MOF (MetaObject Facility).

With strict metamodelling, the relationship between consecutive pairs of levels is always "is-an-instance-of" e.g. the M1 class Person is-an-instance-of the (M2 level) metaclass Class. However, in UML, loose metamodelling is used [27] in which the "is-an-instance-of" relationship is *not* restricted solely to inter-level connections (as in strict metamodelling) but exists *within* a layer e.g. the M2 layer contains both Association and Link (where a link is an instance of an association).

Subtyping (Generalization relationship) can exist in either the M2 or M1 layer e.g. within the M2 layer we find Class as a subtype of Classifier; in the M1 layer we might model SavingsAccount as a subtype of Account (where both are instances of the M2 class Class).

Stereotyping is intended to permit the user effectively to create a subtype in the M2 layer without altering the M2 metamodel but by specifying elements in the M1 (model) level. This leads to the UML definition of a stereotype [27] (p2-65) as "a UML model element that is used to classify (or mark) other UML elements so that they behave in some respects as if there were instances of new "virtual" or "pseudo" metamodel classes". This virtual metalevel (or M2) class is then described as "a specialization of an explicit M_2 class." [2] (p24). This means that a stereotyped class (at the M1 level) is-an-instance-of[2] a specific metasubclass (**Figure 1**). In this example, ControlClass (the metalevel subclass) is a subtype of Class and thus inherits all its metalevel features and relationships. Consequently, since a Class has an attribute (at the M2 level) of isAbstract, then so does the M2 ControlClass. This is manifested in the M1 level Bird class as a class attribute value. Any meta-attribute of ControlClass (but not of Class), such as isCritical, are similarly instantiated in the M1 level class as class level attributes. In addition, since a (M2) Class possesses Features (Attributes and Operations), so does the ControlClass. Thus, the M1 level Bird class instantiates these Features i.e. there are a number of actual attributes and operations of the Bird class e.g. hasFeathers, fly(). Note that these are obtained indirectly from the M2 class Class's connections to M2 classes Attribute and Operation. What comes from the M2 class ControlClass? Anything defined (as a meta-attribute) in ControlClass (but not in Class) is manifested at the class level in Bird. It is not thus possible for the M1 class, here Bird, to obtain *actual* attributes and operations (which would then be available to instances of Bird) from ControlClass directly. Hence these class level attributes, together with the stereotype ≪control≫, are shown in the Name section of the class icon together with any other Constraints applicable to ControlClass but not to Class.

On the other hand, a common, but incorrect, interpretation of stereotyping is that the stereotyped class takes on not only its own features but also those of the stereotype. Typical examples might be ≪control≫ Bird (as shown in Figure 1) and ≪Person≫ Employee wherein one anticipates (wrongly) that an Employee *is-a* Person i.e. that in some way instances of the Employee class also directly acquire features of the Person class [4]. This misinterpretation has been so common that Booch *et al.* [7] were moved to write: "A stereotype is not the same as a parent class in a parent/child generalization relationship" (p80). Consequently, if we want an Employee instance (M0 level) to obtain features relevant to both Employees and Persons, then subtyping at the M1 level is the only UML mechanism. This misunderstanding/confusion regarding UML stereotypes comes from several sources, predominantly:

1. describing stereotypes in terms of *user-defined* subtypes often misses the point that these subtypes are at the M2 level and not at the M1 level

[2]Indeed, Atkinson and Kühne [2] suggest that "the stereotype mechanism represents an alternative way of expressing the instantiation relationship without offering any additional modelling power."

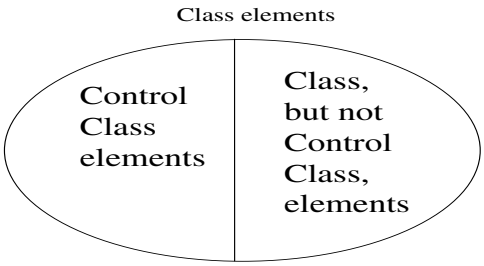

Figure 2. At the M2 level, the "virtual" submetatype of Control Class relates to a fraction of the instances in its parent class (the Class metaclass)

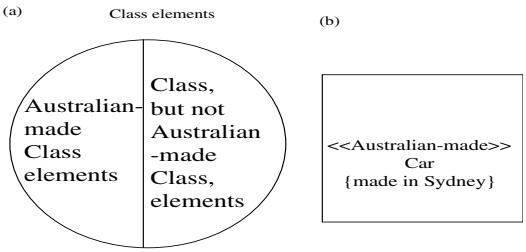

Figure 3. Adapting Figure 2 to a specific example appears correct, but isn't.

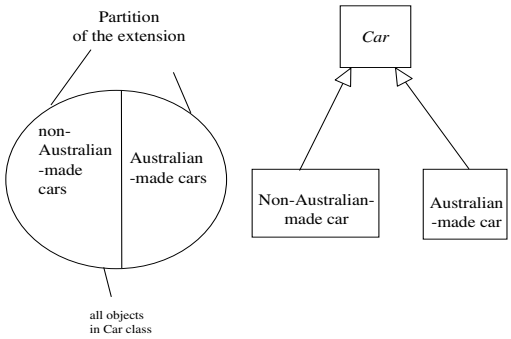

Figure 4. The intended partition for Figure 3. Correct at the M1 level: (a) as a Venn diagram and (b) using generalization relationships

2. the slack use of the phrase "is-a" which has many (mis)interpretations through the UML [4]

Another (mis)interpretation of the stereotype, hinted at in Figure 1, is as a mechanism with which to support multiple, concurrent classification. A stereotype, as shown in Figure 1, effectively creates a partition of the supertype, as shown in **Figure 2**. Thus, there are some Classes which can be described as Control Classes and others which are not. Together, these various classes (viewed here as instances) are members of the metaclass Class. As an example, we consider whether "Australian-made" is an appropriate stereotype. In the context of vehicles, it could be said to apply to *some* cars (automobiles) but not exclusively so i.e. there are Australian-made things which are *not* cars e.g. Australian-made refrigerators. **Figure 3** shows how this might work at the M2 level, in a diagram analogous to Figure 2. Thus if Class (M2) is instantiated as Car (M1), it would be argued that we can create a stereotyped Class of ≪Australian-made≫ Car with an appropriate class level constraint (as shown). Application of the same stereotype to a different class, say Refrigerator, would have similar results — but with the word Car replaced by Refrigerator in Figure 3(b).

All of these interpretations are commonly found in the literature, the third being seemingly closer to their original use in [33] who discusses dual classification schemes — see also ideas of [21] which were very similar although the term stereotype was not used in OOSE/Objectory at that time. Interestingly, the introduction of the word stereotype into object modelling in [33] was in line with the dictionary definition of an "oversimplified conception, opinion, or image", particularly useful in exploratory modelling. Wirfs-Brock [33] notes that the goal is not to introduce stereotypes with the intention that they persist but rather as a transient step in the modelling activity.

The problem here is really one of semantics. In this example, we are trying to model (M1 level) a difference between Australian-made cars and non-Australian-made cars[3]. This is a partition of the instances of a class Car (a.k.a. Automobile) as shown in **Figure 4**(a) which can be represented[4] by subtyping *at the M1 level* (Figure 4(b)). Consequently, Car (and its subtypes) are instances of the M2 class Class *and nothing else* — there is *no* use made of subtyping at the M2 level. What subtyping at the M2 level would offer would be, for instance, the creation of a (meta)class at the M2 level which described "Australian made metalevel Classes" i.e. those metalevel classes introduced by Australian (M2 level) and *not* "Australian-made types" (M1 level) or "Australian-made instances" (M0

[3] The sort of modelling problem that users might feel tempted to create a stereotype for! — as we noted above (Figure 3).

[4] Actually using subtyping in this example doesn't produce a very good model — the use of attributes would probably be more successful than inheritance here.

level). In this case, we would represent this in an M1 model by the ≪Australian-made≫ stereotype.

This leads to the question as to what sort of concepts *are* expressible as (user-defined) subtypes in the M2-level model. Examples of extensions for metaclass Class (some of which are included in the UML, as it happens) are

- AbstractClass (not predefined in the UML)
- PersistentClass (not predefined in the UML)
- Type [although we discuss later, in Section 5, whether this is appropriate]

Other examples in UML exist e.g. for relationships of Usage (a subtype of Dependency) which, in turn, is a subtype of Relationship). These include call, instantiate and create. In all these cases, it is appropriate to create a partition of the M2 class Class or Usage respectively since all of these are "special kinds of" Class and Usage.

Instances of these (at the M1 level) make sense as do instances of the instances (M0); whereas stereotypes such as ≪Australian-made≫ do not. In Figure 3(b), the class essentially delineates a set of instances which conform to that type. The nature of the type is a compound essentially of the class name and the stereotype so it is not a valid question to consider instances which conform to the stereotype but not to the class's type. Thus the idea of an intersection of two classification schemes is not tenable *unless* one of those classifications is a simple Boolean-valued characteristic e.g. Australian-made: Y/N; Persistent: Y/N.

There is then a second, more philosophical question. If a stereotype is "user-defined", how is it possible for the definition of the UML to contain standardized (i.e. methodologist-defined not user-defined) stereotypes? These abound in the UML, being the mainstay of many of the relationships as well as the type/implementation class debate (Section 5). If they are an irrevocable part of the OMG standard, then why not make these part of the metamodel itself. At present, stereotypes are at the user's (M1) level; which leads Atkinson and Kühne [3] to suggest that, rather than the correct definition of metamodelling [1], UML appears to equates the word meta with "predefined"; irrespective of M1/M2 level.

Finally, and in a similar vein, one must challenge why some of these OMG-defined "subtypes" in the UML definition are defined using stereotypes and others by subtypes. For example, consider **Table 1** which summarizes the main relationships in the UML. In the first column are relationships which are all defined as proper subtypes in the metamodel. We see that some have their own representation (dependency, flow, generalization and association) whereas two (extend and include) have stereotype-like annotations. Similarly, in the second column there are four more subtypes visible in the metamodel, two of which have stereotype labels and two of which do not. [9] similarly note that while Interface is a (meta)class in the metamodel, in the notation it is shown by ≪interface≫ attached to a class icon — a notation indistinguishable from a stereotype (yet Interface is *not* a stereotype). From a learnability and teachability viewpoint (Section 7.3), this inconsistency makes it very difficult to undertake the necessary technology transfer to the user community.

Table 1 Relationships and their stereotypes after [19] (Table 2.1)

Subtypes of Relationship in the Metamodel	**Subtypes in the Metamodel (of the Subtypes in first column)**	**Stereotypes**
dependency	abstraction	≪derive≫ ≪realize≫ ≪refine≫ ≪trace≫
	≪bind≫	
	permission	≪access≫ ≪friend≫ ≪import≫
	≪use≫	≪call≫ ≪instantiate≫ ≪create≫ ≪send≫
flow		≪become≫ ≪copy≫
generalization		≪implementation≫
association ≪extend≫ ≪include≫		

Recommendations:

1. Remove stereotype as a user extension mechanism and replace by standard metamodelling with a unified and consistent (across metalevels) notation [1]

2. Retain only "predefined stereotypes" and turn them into full metasubtypes (i.e. subtypes in the M2 model)

3. Whole–Part Relationships

In UML V1.3 there are two relationships, both created by choosing specific metaattribute values on the metaclass AssociationEnd, which purport to be useful for "aggregation". These are called composite aggregation or just composition (black diamond notation) and shared aggregation (white diamond notation). Whilst these are meant to be two kinds of the whole–part (WP) relationship, the basic axioms relevant to all WP relationships (as discussed in [32, 26, 22, 30, 16]) are absent from the UML definition document [27]. Although these may not yet be universally or unequivocally accepted, the set proposed in the more recent papers centres on the need for all WP relationships to include a whole which has emergent and resultant properties and for there to be specific rules regarding asymmetry, antisymmetry and reflexivity.

The current choice of dealing with Composition and Shared Aggregation in the UML metamodel through "AssociationEnd" and its meta-attributes is inadequate. Barbier and Henderson-Sellers [5] clearly demonstrated that this allows the construction of ill-considered WP relationships. This, in particular, results from the total absence of OCL constraints on the "aggregation" attribute of "AssociationEnd". Furthermore, the current definitions of shared and composite aggregation in [27] are both ambiguous and self-contradictory [17]; for example, there are some genuine whole–parts that fall into neither shared nor composite aggregations in current UML and some that fall into both simultaneously. One reason for this confusion is that not only are the basic axioms of WP ignored but the partition chosen for black and white diamond is not uniquely defined. White diamond embodies one partitioning rule and black diamond an orthogonal rule.

In order to clarify "aggregation" in future versions of UML, what is needed is first the realization that there are two basic parts to the problem (and hence to any proposed revision): (i) clarification of a set of axioms which define the semantics of *all* WP relationships; and only then (ii) division of all instances of whole–part relationships into two subsets i.e. a single partition should be identified[5]. These can then be associated with the notations of black and white diamonds (or another symbol such as the Philips screwhead and set membership symbols used in OML [14]).

Obviously, the partitioning needs some secondary characteristic with which to accomplish the separation into two "flavours" of WP. As described in [17], this might be inseparability (a.k.a. invariance) or configurationality — or perhaps even a combination of characteristics which defines 'existence dependency' [31].

As a postscript, although not strictly WP relationships, it should be noted that the Philips screwhead notation introduced in OML [10] as a *specific, semiotically strong* symbol to mean a 'strong' form of WP (actually configurational whole–part), has been borrowed in UML Version 1.3 to mean something totally different. In Version 1.3, this notation was added to model packages contained inside other packages. This is topological inclusion and is NOT a whole–part relationship. The idea of a Philips screwhead, suggested during extensive user field trials, was to connote a tight, parts-joined-to-whole type of relationship. Its use for packages inside packages is, regrettably, cognitive misdirection. As an untested part of UML 1.3, we foresee it causing many user problems. We instead recommend reserving this symbol for configurational whole-part relationships as discussed in [17].

In addition, the black diamond and even the white diamond seems to be used in the UML metamodel in places where the notion of whole-part really does NOT seem to be appropriate (see also Section 6). Often these problems are "fixed" by judicious use of OCL. We believe OCL should be used to express constraints not to fix inadequacies in the metamodel. In other words, once "composite aggregation" is made meaningful, the UML metamodel itself can be rapidly and easily improved significantly.

Recommendations:

1. Clarify the axiomatic set of properties held by *all* whole–part relationships
2. Identify one, possibly two, characteristics to be used to create a clean partition
3. Allocate semiotically useful symbols to the two (or four) kinds of WP Relationship
4. Introduce definitions, semiotics and notation for "topological inclusion"

4. Responsibilities

Responsibilities are high level descriptions of what an object knows, can do and can enforce. They were introduced in [34] in such a way that they have a one to many connection to operations in the interface of a class. The current model of a Responsibility in the UML is as a stereotyped comment[6]. The important

[5] As an alternative, two partitions could be used giving four uniquely defined subsets of the metaclass WholePartRelationship.

[6] Changed from a tagged value in Version 1.1.

conceptual linkage from Responsibility to Operation is ignored in the UML by making a Responsibility merely textual and thus with no semantics. Currently, to mimic the true nature of responsibility à la Wirfs-Brock, two Dependencies need to be manually introduced [6] although this suggestion seems to us inadequate and incorrect (since a Dependency cannot link a Comment to a Classifier). Furthermore, in V1.3, Responsibility and Contract are defined as synonyms. This is incorrect as anyone doing contract-driven programming [25] will be aware. The responsibility is a statement of high-level services (features) supported; the contract are the rules that govern when and how those services can be effected.

The solution is fairly simple but it does require a direct modification to the UML metamodel. A single metaclass must be introduced to sit inbetween Class and Feature with a one to many connection from Class to Responsibility and, again, a one to many connection from Responsibility to Operation (see figure 11 of [20]). Everything else then follows. [A proposal along the lines indicated above has been solicited by the Chair of the RTF from the author and collaborators.]

Recommendations:

1. Introduce a new metaclass (Responsibility) into the M2 model.

5. Types v. Interfaces v. Classes

The current UML model here is a confused mixture of subtypes and stereotypes. The main concept here (in UML 1.3 that is) is Classifier. This is the concept that has a plain notation: the rectangle. Of the several subtypes of Classifier in the metamodel, the two of interest here are Class and Interface. Since these are different concepts, they require a unique notation. The UML choice is to show these subtypes as if they were stereotypes (see also Section 2). So Interface and Class are the basic Classifier rectangle plus a stereotype name — except that an exception is permitted for Class in which this stereotype name (of ≪class≫) is omitted on the grounds of frequency of occurrence of classes in a regular (M1) OO model.

Class itself then has two stereotypes of interest: Type and Implementation Class, again denoted by these names in guillemets. As we noted above, stereotypes can be described as "user-defined submetatypes". It it thus reasonable to depict these with a generalization relationship. That being so, we can assess whether the relationship (between Type and Class, say) as depicted here is truly a generalization/specialization relationship. The UML definition of the Generalization relationship describes (correctly) the notions underpinning polymorphism i.e. that the subtype extends and restricts the supertype and can be dynamically substituted for it. Another way of saying this is that the subtype has all the Features of the supertype and possibly more. Unfortunately, this is not the case with the comparison of Type and Class [20]. A Class in UML has Attributes, Operations and Methods whereas the Type only has Attributes and Operations i.e. the Type is all that the Class is *less* Methods. Similarly, an Interface has no Methods or Attributes, whilst a Classifier has both (both have Operations). These relationships, between Type and Class and between Interface and Classifier, must therefore be implementation inheritance and NOT generalization/specialization. If anything, the relationship should be inverted e.g. a Class is a special form of Type since a Class contains everything that is in the definition of Type — and more.

Of course, seeking an improved definition of the Class/Type/Interface definitions may not be so simple as inverting an inheritance hierarchy — it is hard to argue that semantically *either* a Class is-a-special-kind-of Type *or* a Type is-a-special-kind-of Class. This suggests that neither hierarchy nor inverted hierarchy may be the optimum solution.

An alternative approach to this metamodelling paradox is to consider other relationships between the metaclasses. Since, at least intuitively, a Class is a combination of its external appearance (its interface/type) and its internal detail (Class Implementation) then one of the two models [20] might be useful. In this, different, modelling approach we can think of Class as being an aggregation and its implementation. This obviates the concerns about Classes containing more Features than Types. It could also make Class and Type and Interface peers and remove ImplementationClass from the existing inheritance hierarchy.

A further factor to be considered is the realizes relationship between Classifier and Interface. Since, in this (current V1.3) metamodel, a Classifier may dynamically be a Class or an Interface or a Type or an Implementation Class (or others), then we have a problem. Two necessary consequences of this definition are that:

1. an interface may realize another interface
2. a type may realize an interface

neither of which make any sense. Curiously, in quoting from RUP, [8] state that the only kinds of classifier that can possess interfaces are class, subsystem[7]

[7]Note that subsystem is itself a subclass (of Package and also Classifier). It thus has Features (said to be "constrained to

and component; while at the same time permitting that "an interface can be realized by one or more classifiers" (e.g. type would seem to be valid again here). In contrast, the realizes relationship is between type/interface and implementation *only*, thus obviating the problems caused in V1.3 by the inheritance/stereotype structure as discussed earlier.

The second question implicit in this discussion is the difference between Type and Interface. In UML 1.3 they are clearly different, one being a peer of Class and occurring as a full metatype in the UML metamodel, and the second (type) being more subservient appearing as it does as a stereotype of Class. The rationale for the need for both type and interface is as follows: the public features (or services) offered by a class in an M1 model may need to be subdivided/grouped (possibly overlapping) in order to represent restricted interactions e.g. Group 1 can only be accessed by Class A and Group 2 can only be accessed by Class B. One word is thus needed for this group and one for the full set of features (the sum total of the groups less any common features in the several sets). In OML [10], it was argued that, since many other classifiers had interfaces (e.g. use cases, subsystems), interface would be an appropriate generic name for the full set of visible features. Then the Type represents the restricted type view (a computer science viewpoint) of how that Class might be expected to behave *in that particular context*. When the earlier versions of UML were made available, it became clear that the same two words were being used in the opposite sense, perhaps influenced by the rapid upsurge of Java at that time, which used the word interface to mean a subset of available operations. Of course, if there is only one subgroup then the two terms are synonymous.

If we use the first description, then we can introduce that into the UML metamodel by modelling Interface as an aggregation of Types (**Figure 5**). Types then consist of Features. In the terminology of the latter case above, we have Type as an aggregation of Interface which in turn is an aggregation of Features. If this type of metamodel is used, then we might also consider moving the newly introduced Responsibility metaclass to be connected to Interface/Type (top left hand in Figure 5) rather than to the original supertype of Classifier. All these elements would have to be considered together in discussion of UML Version 2.0 in order to ensure consistency.

Thus, the question becomes (a) is it worthwhile retaining two concepts, called type and interface and, if so, which is the best use of this pair of terms or (b) would it be better to use one word, probably for the subgroup notion, and replace the second term by a phrase describing it as a special case of the other (adopted) term? Such a merger of terms (between interface and type) would seem to accord with the stated aims of the UML RTF [23] of "slimming down" the UML in Version 2.0.

be Operations and Receptions" [27] (p2-174)). Package, on the other hand, is not a Classifier, inheriting multiply from Namespace and GeneralizableElement

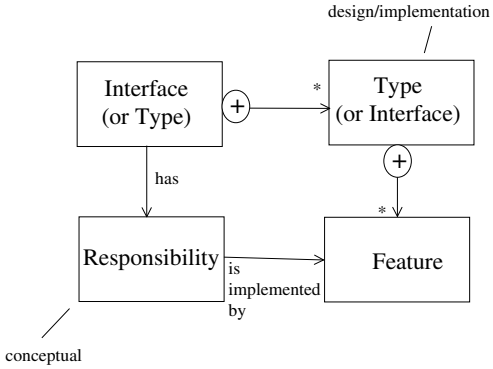

Figure 5. A metamodel fragment to show how Whole-Part relationships may be used to simplify the Type/Interface relationship

Recommendation:

1. Totally redesign this part of the metamodel. Eliminate most of the inheritance-related relationships (Generalization and Stereotypes) and consider an architecture of this part of the UML metamodel which uses WPRelationship and Associations.
2. For the type/interface dilemma, there seem to be (at least) two possibilities:
   a) Keep one word (type or interface) to represent the full extent of the external set of features and the other word to represent a subset of these. The current definitions of UML and OML take opposite views here.
   b) Use only one of these concepts and consider (prove/show) the other as a special case.

Both of these proposals could be successful — at least in the isolation of this metamodel fragment.

6. The (Mis)Use of UML in the UML Metamodel

The UML metamodel is described by using the UML notation. This circular use can work well but relies on

the watertight and correct definitions of all the core elements in UML that are used in the metamodel definition. There seems to be no constraint on which parts of the UML notation can be used in the metamodel and, unfortunately, some notation is used which is not adequately defined elsewhere in the UML documentation [27].

The first of these (mis)uses occurs by the extensive use of black and white diamond notations in the diagrams of the metamodel. The metamodel needs to be semantically tight. Since we have seen that the meaning of black and white diamonds in UML V1.3 is both ambiguous and self-contradictory [17], then those portions of the metamodel which utilize black and white diamonds (almost all of it!) must likewise be ambiguous and of low quality. It is even challengeable that some of the parts of the metamodel with black or white diamond annotations are even whole–part; another problem with the deficiency of a set of basic axioms defining WP within the UML.

Granted, some of the ambiguity can be removed by judicious use of OCL — but the use of OCL to fix errors in the metamodel is NOT to be encouraged (hopefully not even to be permitted!). Thus, it is crucial to insist on watertight definitions of the various partitions of "aggregation" *before* the associated notations are used in the metamodel component of the UML's definition. Clarification of these semantic issues will quickly make the metamodel more usable and meaningful.

For example, several of the black diamond composition relationships in the core (static) metamodel of UML Version 1.3 require serious reconsideration. We cannot here state definitively whether these particular examples would survive scrutiny since the results depend directly on the selected definition and partitioning of the WPRelationship — as discussed in Section 3. In the current version, Classifier is shown as a composition of Feature; Association as a composition of AssociationEnd; and AssociationEnd as a composition of Attribute. Certainly an AssociationEnd (for instance) "has" Attribute(s) but naming an association, even at the metalevel, as "has-a" does *not* automatically transform that relationship into an aggregation. Very few "has-a" relationships have any connection to aggregation/composition/WP, relating more to ownership — as in the phrase "Peter has a car". As a second example, consider the use case metamodel in which a use case is said to consist of attributes and operations and for a namespace's constituents to be use cases. Association is clearly valid, but the "elevation" to WPRelationship is dubious and would need to be well argued against the primary and secondary characteristics of the Whole–Part Relationship as outlined in Section 3 and discussed in detail elsewhere [16].

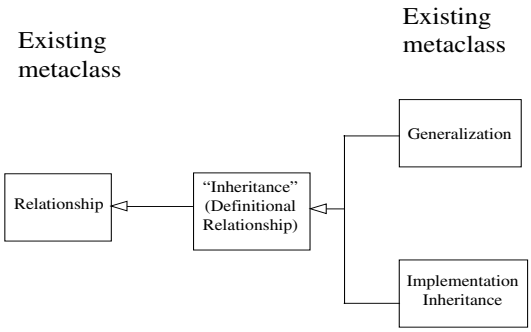

Figure 6. A new partition for the "inheritance" family of relationships

A second problem occurs with respect to the use of the white-headed generalization arrow. This means "is a kind of" i.e. subtyping. It supports polymorphic substitution because the subtype (which is being substituted) supports all the Features of the parent (and more). This is NOT the case in various parts of the UML metamodel. For instance, we have noted that a message sent to a Class, when dynamically routed to the corresponding Type, may be unsuccessful since the Type has no methods, This is compounded by the incorrect definitions of the predefined stereotypes of the generalization relationship. Called ≪implementation≫, this stereotype was added to permit the modelling of implementation inheritance relationships. Unfortunately, because it is a stereotype, it can be regarded as containing a subset of all the instances of its parent type. In other words, the current UML model is that Implementation Inheritance is a special kind of Generalization. Implementation Inheritance is (by this definition) all that Generalization is and probably more. This is blatantly untrue since it does not support the interface or substitutability [20]. Of all the "inheritance relationships", one could model them by a partition which divided the set into two portions: those representing specialization/specification inheritance (generalization) and those representing implementation inheritance. These would preferably be subtypes (rather than stereotypes) in the new metamodel (**Figure 6**).

Recommendations:

1. Following adoption of better definitions for black diamond and a reconsideration of the partitioning of various inheritance types, each use of them in the current metamodel must be scrutinized.

7. Some Other Concerns

Other issues discussed briefly in this section include arguments over default directionality of associations and the relationship of an association to a dependency; the support for seamlessness; and the general observation that the UML uses a notation that supports expressibility but is not in itself expressive leading to a brief discussion on issues raised in UML education.

7.1. Relationships

The major relationships in UML Version 1.3 are Generalization, Dependency, Association and Link, which are normally seen in class diagrams; Include and Extend, for use on use case diagrams and Transitions on Statechart Diagrams (see figure 4 of [20]). Dependency has four subtypes each of which has several predefined stereotypes. From the Association metaclass, it is possible effectively to create three kinds of association (normal association, shared aggregation and composite aggregation) by setting different values to the metaattribute called aggregation:AggregationKind on the metaclass AssociationEnd. In passing we note that Link and Transition are not kinds of relationship but this would seem to be a "typographical error" in the UML, particularly since Link is an instance of Association although it may be this very fact that leads it to be treated differently and symptomatic of the "loose metamodelling" of the current version of UML.

When this structure was made available late 1998/early 1999 in draft form and June 1999 in final form, it created extensive discussions about the correspondence between Dependency and Association. Arguments ranged supporting two opposite statements: A Dependency is-a-kind-of Association and Association is-a-kind-of Dependency. Neither is the case at present (other than a common parent in the metaclass Relationship). [29] discusses this issue and ends up recommending that users view Dependency as just another viewpoint on an Association. While not evident in the UML metamodel, we would concur with this statement — it *is* a good way of seeing it [14] and, furthermore, it may be profitable to view all relationships as dependencies [20] i.e. to merge these two M2 level concepts with the "association" flavour predominating in "analysis" and, probably, "dependency" in design.

An opportunity afforded by a major revision such as that proposed for Version 2.0 is to consider enhancement of the relationship hierarchy based on the existing Relationship metatype. **Figure 7** shows how a clean, four-stranded inheritance hierarchy may work — as implemented in OML [10] — and offer a useful base from

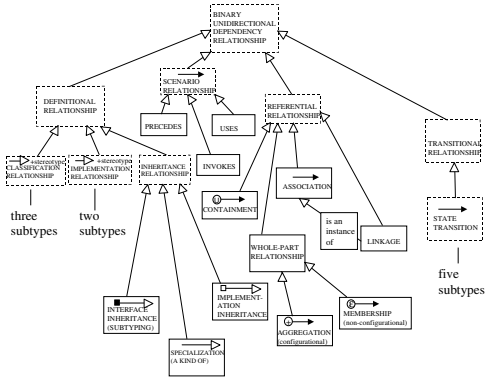

Figure 7. Proposal for a four-strand hierarchy of relationships (based on that originally proposed in OML [10])

which to propose a revision to the UML metamodel. This structure offers flexibility (e.g. other types of inheritance can easily be added) and clearly differentiates between those relationships which are definitional and those which are clearly referential. It also fuses together the ideas of Dependency and Association, seeing the former as the dynamic version of the latter (as noted earlier). In the spirit of slimming down the UML [24], we suggest this type of rationalization should be considered. To do this, we will focus our attention on the definitional and referential branches i.e. those used in the class diagram.

We would thus propose creating two branches under Relationship: ReferentialRelationship and DefinitionalRelationship; the latter could be also called "inheritance" in a very loose sense. On the ReferentialRelationship branch we would have not only Association+Dependency from the revised Relationship hierarchy but also introduce a full metaclass for the various forms of WholePartRelationship. At the same time, we might also consider a special sort of referential relationship known as Containment (as in a container with contents). Finally, we might take a more drastic restructuring move by arguing (and this is contentious) that the notion of Dependency is in fact generic to both ReferentialRelationship and to DefinitionalRelationship and thus that characteristic can be taken out of the "Association+Dependency" class and placed higher in the hierarchy, thus changing Relationship itself to DependencyRelationship (**Figure 8**).

Another concern relates primarily to the Association metatype. It should be noted that there is some inconsistency with respect to directionality. By choosing bidirectionality as the default for associations, rep-

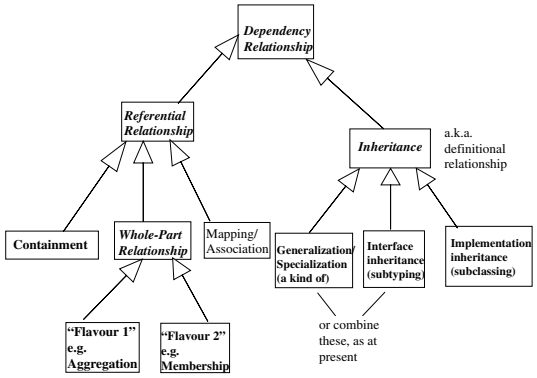

Figure 8. Proposed relationship hierarchy for a future version of UML based on a rationalization of the existing UML Relationship hierarchy using ideas from Figures 6 and 7 (modified from [15])

resented by undirected arrows, there are two consequences: (i) it not possible to unambiguously draw a To-Be-Decided arc in UML (since these are confounded in the UML); and (ii) although Association is bidirectional, all its subtypes and stereotypes are *uni*-directional — which again negates the ideas of the subtype being an extension of its parent (a subtype). In other words, since bidirectionality is more feature-rich than unidirectionality, then any relationship possessing bidirectionality should be metamodelled as the subtype with the unidirectional concept as the parent — the opposite of the current UML.

Furthermore, it has been shown [13] that bidirectional associations violate encapsulation. A bidirectional association is actually a pair of unidirectional associations between the power types with appropriate referential integrity constraints. This suggests that unidirectional associations should be the default, rather than the current bidirectionality.

Recommendations:

1. Evaluate the efficacy of merging Dependency and Association
2. Make all Dependency Relationships one way (as default)
3. Differentiate clearly, in notation and metamodel, between definitional and referential relationships (which could be new subtypes in the metamodel)
4. Possibly move Dependency characteristic to top level in the hierarchy by changing Relationship to DependencyRelationship

7.2. Support for Seamlessness

OO strongly supports the idea of a seamless transition from requirements through analysis and design to code. It is generally agreed that requirements and analysis focus on the external view of an "object". This means either the interface, the type or the specification (depending upon agreed definitions). In design and code, the emphasis shifts somewhat to the internal descriptions and thus to the class and finally to the object at runtime (or possibly in detailed design, particularly for real-time modelling situations). This means that, using UML, one should use the Type or Interface symbol (a stereotyped Classifier icon) in analysis and then switch to the Class (an unstereotyped Classifier icon) in design, typically types and classes in programming and then objects (in which an underscore is added to the name on the Classifier icon). This introduces additionally the need to define what is meant by analysis and design and when these "phases" end. It goes against the idea of seamlessness, iterative development and reverse engineering. What is ideally needed is a symbol for a concrete metatype which is the parent of Class, Type, Interface, Object/Instance. This could be Classifier but (a) Classifier is abstract not concrete and (b) the symbol commonly used in UML is in fact the Class not the Classifier. We recommend the use of an additional symbol for this parent (now concrete) metatype. Currently, constant use of the Class icon throughout the *whole* of the development amounts to cognitive misdirection. The opportunity of a major revision might be the time to re-evaluate the linkages between the metatype hierarchy (per se) as well as the linkages from this to the notation.

Recommendations:

1. Make Classifier a concrete, not an abstract, class in the UML metamodel
2. Introduce a new notation for Classifier; or else insist on use of ≪class≫ stereotype annotation for classes (M1)

7.3. Expressiveness and Learnability

Expressiveness relates to the degree that (here) a notation is able to easily and effectively convey concepts, particularly to the novice user. A language is expressive if it is effective and simple to "get a message across". In contrast, expressibility is the ability to express things. In natural language, as an example, it is perfectly possible to use most natural languages to express most things. However, it is easier in some than

in others. As an example, consider the wide variety of tenses of verb in English. This permits easy expressive use of a wide variety of (often subtle) differences not possible easily in other languages. In Chinese, for instance, there is no direct support for the future tense of a verb; you have to add phrases such as "tomorrow", "in two weeks time" and so on but still use the present tense of the verb. This is clumsy to an English ear that has a future tense available. In this context, both languages will support the expression but English (in this example) is more expressive than Chinese.

UML can be considered to be more like Chinese in that, while being perfectly possible to express almost anything, many are convoluted. Take the way suggested above to express a Responsibility by manually inserting two dependency relationships compared to the ideas expressed above where a Responsibility is semantically linked (automatically) to operations in the interface. There are many other examples[8].

Finally, from our experience in extensive teaching of the UML, that there are a few technical inconsistencies which confuse the novice user of UML. For example, a number of students[9] have asked why, since there is an exterior/interior pairing of operation and method (which is nice), there is nothing equivalent for the static side of objects (attributes) e.g. one could have logical attributes in the interface and physical ones inside. The outside would then match to the old idea of queries and commands. We also noted earlier (Section 2) the inconsistent use of stereotype symbols for actual metaclasses. We also note anecdotal evidence from UML novices that the language is found to be large and cumbersome to learn. This is largely because it eschews well-established usability and semiotic theories and experiences, being largely market-driven in its inception and development. This leads to the need for novices to indulge in extensive rote learning, an aspect of the UML which is already leading to some companies dismissing it as inappropriate or useless.

Recommendations:

It is hard to make specific recommendations here since the UML development team has already eschewed any need to consider semiotics or learnability being driven by more pragmatic considerations such as market share. However, if/when new concepts, such as some of those discussed in this paper, are introduced into the next version(s) of the UML, the issue of learnability and semiotics should be put on the agenda.

[8] The difference between expressiveness and expressibility in the context of object modelling and the UML was first pointed out to me by Richard Riehle at the TOOLS Pacific Conference in November 1999.

[9] Martin Fowler makes the same point, we note [11, 12].

8. Summary and Conclusions

We have identified a number of problems with the current version of the Unified Modeling Language (UML) and discussed the underlying theory relevant to both the problem and some potential solutions. In particular, we have investigated stereotypes, whole–part relationships, types, interfaces and classes. We have also looked some aspects of how the UML is itself (mis)used in its own metamodel and drawn the discussion to a close by evaluating relationships more generally and some aspects of usability and learnability. The suggestions are made not critically but constructively in order to improve future versions of the OMG's modelling language.

9. Acknowledgements

This paper was initially based on ideas submitted to the OMG UML RTF's RFI for Version 2.0. Signatories to that OMG document (ad/00-12-11), whose contribution is hereby acknowledged, were Scott Ambler, Franck Barbier, Donald Firesmith, Ian Graham, Bernhard Rumpe, Clemens Syzperski, Richard Thomas, Alan Cameron Wills, Bhuvan Unhelkar and the Kerry Raymond representing the DSTC (a contributing OMG member). Thanks also to Colin Atkinson for providing preprints of his papers submitted to ECOOP Workshop and ≪UML≫2000. Thanks also go to Colin Atkinson, Franck Barbier and Thomas Kühne for helpful and constructive comments on an earlier draft of this paper.

This is Contribution Number 00/9 of the Centre for Object Technology Applications and Research (COTAR).

References

[1] Atkinson, C., 1999, Supporting and applying the UML conceptual framework, *The Unified Modeling Language. ≪UML≫'98: Beyond the Notation* (eds. J. Bézivin and P.-A. Muller), LNCS 1618, Springer–Verlag, Berlin, 21–36

[2] Atkinson, C. and Kühne, T., 2000a, Meta-level independent modelling, ECOOP Workshop, France, June 2000 (in press)

[3] Atkinson, C. and Kühne, T., 2000b, Strict profiles: why and how, ≪UML≫2000, York, UK, September 2000 (in press)

[4] Atkinson, C., Kühne, T. and Henderson-Sellers, B., 2000, To meta or not to meta — that is the question, *JOOP* (in press)

[5] Barbier, F. and Henderson-Sellers, B., 2000, The whole–part relationship in object modelling: a definition in cOlOr, *Inf. Soft. Technol.* (in press)

[6] Booch, G., 1998, personal communication

[7] Booch, G., Rumbaugh, J. and Jacobson, I., 1999, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, USA, 482pp

[8] Christerson, M. and Undén, N., 1999, Modeling large systems, *ROSE Architect,* **Fall 1999** obtainable from http://www.rosearchitect.com/mag/archives/9904/magnus. shtml

[9] Dykman, N., Griss, M. and Kessler, R., 1999, Nine suggestions for improving UML extensibility, *≪UML≫'99 — The Unified Modeling Language. Beyond the Standard* (eds. R. France and B. Rumpe), Lecture Notes in Computer Science 1723, Springer–Verlag, Berlin, 236–248

[10] Firesmith, D., Henderson-Sellers, B. and Graham, I., 1997, *OPEN Modeling Language (OML) Reference Manual*, SIGS Books, New York, 276pp; Cambridge University Press, New York, 1998

[11] Fowler, M. with Scott, K., 1997, *UML Distilled. Applying the standard object modeling language*, Addison-Wesley, Reading, MA, 179pp

[12] Fowler, M. with Scott, K., 2000, *UML Distilled Second Edition. A Brief Guide to the Standard Object Modeling Language*, Addison-Wesley, Reading, MA, 185pp

[13] Graham, I.M., Bischof, J. and Henderson-Sellers, B., 1997, Associations considered a bad thing, *J. Obj.-Oriented Programming,* **9(9)**, 41–48

[14] Henderson-Sellers, B., 1998, Towards the formalization of relationships for object modelling, *TOOLS 25* (eds. C. Mingins, R. Duke and B. Meyer), IEEE Computer Society, Los Alamitos, CA, 267–284

[15] Henderson-Sellers, B., 1999, OML: proposals to enhance UML, *The Unified Modeling Language. ≪UML≫'98: Beyond the Notation* (eds. J. Bézivin and P.-A. Muller), LNCS 1618, Springer–Verlag, Berlin, 349–364

[16] Henderson-Sellers, B. and Barbier, F., 1999, What is this thing called aggregation?, *TOOLS29* (eds. R. Mitchell, A.C. Wills, J. Bosch and B. Meyer), IEEE Computer Society Press, 216–230

[17] Henderson-Sellers, B. and Barbier, F., 1999, Black and white diamonds, *≪UML≫'99 — The Unified Modeling Language. Beyond the Standard* (eds. R. France and B. Rumpe), Lecture Notes in Computer Science 1723, Springer–Verlag, Berlin, 550–565

[18] Henderson-Sellers, B. and Mellor, S., 1999, Tailoring process-focussed OO methods, *JOOP/ROAD*, **12(4)**, 40–44, 59

[19] Henderson-Sellers, B. and Unhelkar, B., 2000, *OPEN Modelling with UML*, Addison-Wesley, Harlow, England, 245pp

[20] Henderson-Sellers, B., Atkinson, C. and Firesmith, D.G., 1999, Viewing the OML as a variant of the UML, *≪UML≫'99 — The Unified Modeling Language. Beyond the Standard* (eds. R. France and B. Rumpe), Lecture Notes in Computer Science 1723, Springer–Verlag, Berlin, 49–66

[21] Jacobson, I., M. Christerson, P. Jonsson and G. Övergaard, 1992, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, New York, NY, USA, 524pp

[22] Kilov, H. and Ross, J., 1994, *Information Modeling. An Object-Oriented Approach*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 268pp

[23] Kobryn, C., 1999, UML 2001: a standardization odyssey, *Comm. ACM,* **42(10)**, 29–37

[24] Kobryn, C., Chonoles, M.J., Cook, S., D'Souza, D., Iyengar, S. and Ramackers, G., 1999, UML2.0 architectural crossroads: sculpting or mudpacking? *≪UML≫'99 — The Unified Modeling Language. Beyond the Standard* (eds. R. France and B. Rumpe), Lecture Notes in Computer Science 1723, Springer–Verlag, Berlin, 131–139

[25] Meyer, B., 1988, *Object-Oriented Software Construction*, Prentice Hall, Hemel Hempstead, UK, 534pp

[26] Odell, J.J., 1994, Six different kinds of composition, *J. Obj.-Oriented Prog.,* **6(8)**, 10–15

[27] OMG, 1999, OMG Unified Modeling Language Specification, Version 1.3, June 1999, OMG document ad/99-06-09

[28] OMG, 1999, Response against the UML 2.0 RFP, document ad/99-12-11

[29] Rumbaugh, J., 1998, Depending on collaborations: dependencies as contextual associations, *JOOP,* **11(4)**, 5, 8–9

[30] Saksena, M., France, R.B. and Larrondo-Petrie, M.M., 1998, A characterization of aggregation, **in** *OOIS'98* (eds. C. Rolland and G. Grosz), Springer, 11–19

[31] Snoeck, M. and Dedene, G., 1998, Existence dependency — the key to semantic integrity between structural and behavioral aspects of object types, *IEEE Trans. Software Eng.,* **24(4)**, 233–251

[32] Winston, M.E., Chaffin, R. and Herrmann, D., 1987, A taxonomy of part–whole relations, *Cognitive Science,* **11**, 417–444

[33] Wirfs-Brock, R., 1994, Adding to your conceptual toolkit: what's important about responsibility-driven design?, *Report on Object Analysis and Design,* **1(2)**, 39–41

[34] Wirfs-Brock, R. and Wilkerson, B., 1989, Object-oriented design: a responsibility-driven approach, *Proc. OOPSLA'89, SigPlan Notices,* **24(10)**, 71–76