Rearchitecting the UML Infrastructure

COLIN ATKINSON
University of Mannheim
and
THOMAS KÜHNE
Darmstadt University of Technology

Metamodeling is one of the core foundations of computer-automated multiparadigm modeling. However, there is currently little agreement about what form the required metamodeling approach should take and precisely what role metamodels should play. This article addresses the problem by first describing some fundamental problems in the industry's leading metamodeling technology, the UML framework, and then explaining how this framework could be rearchitected to overcome these problems. Three main issues are identified in the current framework: the *dual classification* problem arising from the need to capture both the logical and physical classification of model elements, the *class/object duality* problem arising from the need to capture both the classlike and objectlike facets of some model elements, and the *replication of concepts* problem arising from the need to define certain concepts multiple times. Three main proposals for rearchitecting the UML framework to overcome these problems are then presented: the separation of logical and physical classification dimensions, the unification of the class and object facets of model elements, and the enhancement of the instantiation mechanism to allow definitions to transcend multiple levels. The article concludes with a discussion of other practical issues involved in rearchitecting the UML modeling framework in the proposed way.

Categories and Subject Descriptors: D.1.5 [**Programming Techniques**]: Object-Oriented Programming; D.2.11 [**Software Engineering**]; D.3.3 [**Programming Languages**]: Language Constructs and Features—*classes and Objects*; H.1.1 [**Models and Principles**]: Systems and Information Theory

General Terms: Languages, Standardization, Design

Additional Key Words and Phrases: Metamodeling, classification, unified modeling language, UML infrastructure, classification dimensions, strict metamodeling, deep instantiation

1. INTRODUCTION

Computer-automated multiparadigm modeling is based on the premise of giving modelers the most appropriate modeling abstractions for their particular problem domain, and automatically transforming the resulting models into solution abstractions in the selected implementation platform. This implies the

Authors' addresses: C. Atkinson, University of Mannheim, Germany; T. Kühne, FG Metamodellierung, FB Informatik, TU Darmstadt, Wilhelminenstr. 7, 64283 Darmstadt, Germany; email: kuehne@informatik.tu-darmstadt.de.

need for a general modeling framework that allows users to not only define the modeling abstractions that best suit their needs but also to define their preferred mappings of these abstractions to other implementation-specific concepts. The software industry is currently trying to address this need in the new version of the Unified Modeling Language (UML2.0) and the related Model Driven Architecture (MDA) initiative.

If the UML is to meet this challenge it needs to progress from being a single language to a family of languages based upon a common core set of concepts. This will allow varying levels of features to be provided to different kinds of users without creating a collection of unrelated specialized languages. The need for such an extensible language framework is a central part of the request for proposals (RFPs) for the new UML version 2.0 scheduled for standardization in the latter part of 2003. In the RFPs, the definition of the UML is organized into two parts: the infrastructure, which describes the overall framework within which UML modeling is performed and the superstructure which populates this framework with modeling concepts that constitute the UML modeling language.

Although the need for an extensible modeling framework is generally recognized, there is much less consensus on what form this should take. Existing versions of the UML are based on a four-level metamodeling hierarchy, in which each level (except the top) is presented as being an "instance of" the level above. The basic idea is to extend the traditional object-oriented modeling approach to multiple levels so that UML concepts can themselves be described as instances of a UML-like model. Although the basic approach is sound and the presented linear modeling hierarchy is superficially appealing, over time a number of subtle problems have been uncovered that are not immediately apparent. Many proposed solutions have been suggested, but unfortunately many of these introduce different problems of their own.

In this article we trace the evolution of the UML modeling framework and explain the various interpretations of metamodeling that have been applied and proposed. We focus in particular on the initial motivations for the strictness doctrine, and explain why different interpretations of metamodeling at first sight seem at odds with this principle. We then argue that the key to resolving these apparent conflicts is to recognize two fundamentally different forms of classification. After pointing out the implications of this distinction for the definition of variants of the UML we show how the doctrine of strict metamodeling indeed has a valuable role to play. We then summarize the consequences of the resulting modeling framework and propose further simplifications based on a unification of modeling elements and the concept of deep instantiation. Finally, we discuss how the improved modeling framework relates to aspects of the UML infrastructure.

2. THE EXISTING UML MODELING FRAMEWORK

Although some adjustments to the UML metamodeling framework have been made during the UML's evolution, the core ideas have basically remained unchanged from versions 1.0 to 1.4 [OMG 2001]. In this section we describe these

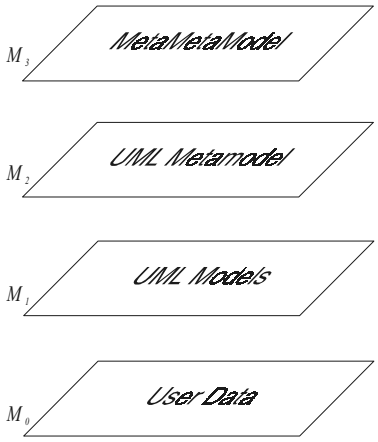

Fig. 1. The four-level metamodeling architecture.

core ideas and provide some historical context behind their development and motivation. Note that for UML versions 1.0 to 1.4 we use the term "metamodeling framework" to refer to the conceptual framework in which UML models reside. However, in accordance with the UML 2.0 RFPs, we use the term infrastructure when referring to the UML 2.0 conceptual framework.

2.1 Linear Model Hierarchy

Although the UML's approach to metamodeling has had several important influences [Henderson-Sellers and Bulthuis 1997], the overall architecture of the UML modeling framework is most heavily influenced by the CDIF [1994] standard developed by a consortium of CASE tool vendors. When the initial version of the UML was being standardized, the CDIF standard was one of the most mature approaches to metamodeling and provided an elegant and flexible approach for supporting the interchange of data (e.g., models) between different CASE tools. It achieved this by describing the different modeling languages used to create specific user models in terms of a single fixed, core model (i.e., a meta-metamodel). Any tool that understood this core model could read the description of a specific modeling language, and thus understand any models written in that language. Moreover, since conceptual models are a powerful description technique, it made sense to view both the definition of a particular modeling language, and the language used to create this definition, as models in their own right. The result was a linear arrangement of models of the form depicted in Figure 1, where the meta-metamodel is the core from which the descriptions of specific modeling languages (i.e., specific language metamodels) are created. Specific models are then viewed as instances of these language metamodels, and user data, as they are known in the CDIF approach, are viewed as an instance of a model. The total number of levels was four because this is sufficient for the purpose of achieving CASE tool interoperability.

A simple example of how this approach is intended to work is illustrated in Figure 2. The example (which is elaborated in subsequent sections) relates to

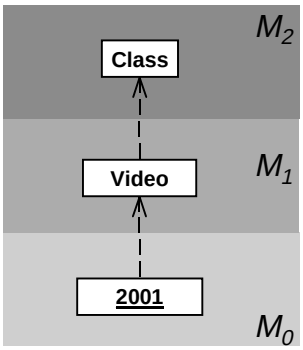

Fig. 2. Extending the two levels of instantiation.

products that might be sold by a shop. An example of such a product is the video "2001: A Space Odyssey" conceptually residing at the bottom level M_0 (i.e., representing user data). Such objects are viewed as being "instances of"[1] classes in a model at the M_1-level above. In the example, 2001 is modeled as an instance of Video. Model elements at level M_1, such as Video, are regarded as instances of metamodel elements from the M_2-level above, such as Class, and these in turn are regarded as instances of meta-metamodel elements (not shown in this diagram).

2.2 Strict Metamodeling

Although conceptually appealing, on closer inspection this simple approach exhibits a number of fundamental problems (at least as originally formulated). One of the most fundamental shortcomings is that there is no precise definition of the "instance-of" relationship. The UML documentation simply states that "A model is an instance of a metamodel" and "A metamodel is an instance of a meta-metamodel." [OMG 2001], but the precise meaning of the instance-of relationship is not defined. This vagueness also made it easy to gloss over the fact that there may be different flavors of the instance-of relationship, depending on which model levels (M_3 to M_0) they connect. The existing version of the UML modeling framework [OMG 2001] takes a very liberal and pragmatic approach in interpreting levels and instance-of links, with the result that the model levels are used as little more than a packaging mechanism.

An early attempt to introduce some rigor into the use and organization of the level hierarchy was the formulation of the strict metamodeling principle [Atkinson and Kühne 2000]. Strict metamodeling is based on the tenet that if a model A is an instance-of another model B then every element of A must be an instance-of some element in B. In other words, it interprets the instance-of relationship at the granularity of individual model elements, as illustrated schematically in Figure 3. Strict metamodeling therefore holds that levels are formed purely by instance-of relationships, not by any other unstated

[1]In this article, "instance-of" relationships of all flavors are always represented as dashed arrows (dependency links) as illustrated.

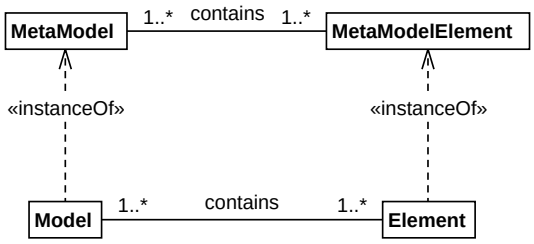

Fig. 3. Strict metamodeling.

criteria. It also mandates that levels have strict boundaries that may not be crossed by relationships other than instance-of relationships. Furthermore, every instance-of relationship must cross exactly one metalevel boundary to an immediately adjacent level. The precise definition of the concept is as follows.

Strict Metamodeling: *In an n-level modeling architecture,* $M_0, M_1, \ldots, M_{n-1}$, *every element of an* M_m*-level model must be an* instance-of *exactly one element of an* M_{m+1}*-level model, for all* $0 \leq m < n - 1$*, and any relationship other than the* instance-of *relationship between two elements X and Y implies that level*$(X) =$ *level(Y).*

This definition deliberately rules out the top level in a hierarchy of levels, since in practice one often wants to terminate the hierarchy of metalevels. A common approach is to model the top level so that its elements can be viewed as instances of themselves. In terms of the model-level instance-of relationship, this is described as a model being an instance of itself.

Although the last rule, banning relationships other than instance-of relationships from crossing metalevel boundaries, might seem like an artificial restriction, it is of the utmost importance for maintaining the concept of a multilevel framework. Without it, the multilevel hierarchy would collapse into a single level [Atkinson and Kühne 2001a]. For instance, relationships between M_0-level entities, called links, are normally distinguished from relationships between M_1-level entities, called associations. If a relationship (other than an instance-of relationship) were allowed to connect an M_0-level entity to an M_1-level entity should it be regarded as a link or an association? Its meaning would be undefined within the four-level framework. Trying to define it—assuming the above example—at level M_2 would destroy the idea that a model can be completely understood as an instance of the model one level higher up.

Since strictness appeared to provide a foundation upon which a sound metamodeling hierarchy could be erected, and offered a discipline for the development of metamodels, adherence to strictness is recommended in the more recent versions of the UML [Kobryn 2001; OMG 2001].

2.3 Stereotypes

Adding to the vagueness of the instance-of relationship was the introduction of an additional mechanism for defining metainformation: the stereotype

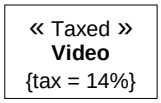

Fig. 4. Application of a stereotype.

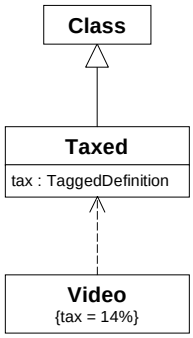

Fig. 5. Effect of a stereotype application.

mechanism. Figure 4 illustrates the classification of Video as a taxed product type. The intent of this example is to indicate that Video is a particular kind of class (a taxed product type) which has a particular property associated with it (a tax value). In other words, videos are products that are taxed at the rate of 14%. Since this value applies to *all* videos, the intent is to attach this value to the class rather than to specific instances of the class such as 2001. In principle there is no need for a new mechanism since its effect (i.e., the classification of classes) could have been achieved by allowing users to extend the UML metamodel. The same logical effect as that illustrated in Figure 4 can be achieved by adding a Taxed concept to the UML metamodel as depicted in Figure 5. In fact, Figure 5 can be interpreted as describing the meaning of this stereotype.

According to the hierarchy shown in Figure 2 the level above user classes classifies or stereotypes classes. Figure 5 also makes it evident that tagged values (such as tax) are best understood as class slots instantiated from corresponding metaclass attributes.

The argument against such user-accessible metamodeling in existing versions of the UML framework is that it demands a "meta" capability in CASE tools. In other words, tools would no longer be able to assume a fixed metamodel that could be hardcoded into the software but would need to treat this as mutable data in their own right. Unfortunately, when the UML was originally developed only a few tools had the capability to change their behavior based on the latest state of the metamodel, so to accommodate the majority of tools that were not able to support metamodeling directly, "lightweight" ways of simulating changes to the metamodel were added to the language. The resulting features have become known as the "stereotype" and "tagged value" mechanisms as illustrated in Figure 4.

Since their introduction, stereotypes have caused a lot of debate among experts as to how to correctly interpret and use them [Henderson-Sellers 2002].

Fig. 6. A stereotype classifying instances.

A significant number of UML modelers embraced stereotypes as an alternative way of classifying instances (rather than classes). Figure 6 shows a typical stereotype application, in which the stereotype is used to make a statement about the instances of a class as opposed to making a statement about the class itself. A Description *instance* physically exists as an HTML page but the *class* Description is certainly not an HTML page.

Not only did the introduction of an additional classification mechanism (stereotyping) compound the existing vagueness of instantiation semantics, it introduced further confusion about its proper role and interpretation in terms of the metamodeling framework.

2.3.1 *The Role of the MOF.* In addition to CDIF, the so-called Meta Object Facility (MOF) has also played an important role in shaping the UML modeling framework. Its origin was in the OMG's Object Management Architecture (OMA) associated with its important middleware standard, CORBA. The OMA envisaged a set of standard objects known as Common Facilities, which would be available in all CORBA-compliant distributed systems to provide frequently needed services through standard interfaces. One of the envisaged Common Facilities was the Meta Object Facility whose purpose was to provide a standard way of accessing run-time metainformation about objects within the system. Thus the MOF was originally conceived as a distributed system service providing access to metainformation via standardized interfaces. Technically, the original goal of the MOF was therefore to provide a set of reflection interfaces.

With this background it is clear that metamodeling, as such, was not a concern in the design of the original MOF concept. However, because the MOF could be used to not only access UML models but also the UML language definition itself (i.e., the UML metamodel) there appeared to be value in integrating the MOF concept with the CDIF-inspired metamodeling framework described above. Unfortunately this eventually led to the MOF being understood solely as the meta-metamodel at the M_3-level and its original purpose and emphasis on reflection facilities became lost.

3. PROBLEMS WITH THE EXISTING UML FRAMEWORK

The concepts described in the previous section still characterize the UML modeling framework up to and including version 1.4. Although it has served its basic role as the backbone for the UML, experience and research have shown that it does not stand up well to close scrutiny. Adhering to strictness while maintaining a linear modeling hierarchy has proved to be particularly problematic. In this section we elaborate upon some of the main problems with the UML's existing modeling framework and discuss some of the solutions that have been proposed by the research community.

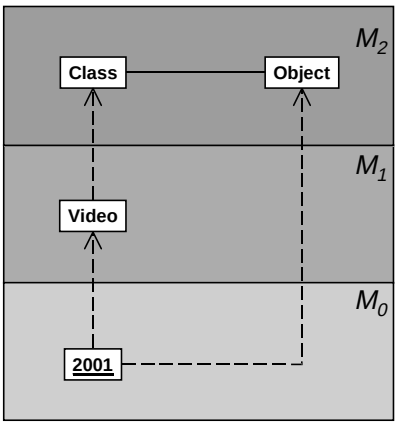

Fig. 7. Multiple classification in the current UML framework.

3.1 Multiple Classification

The core problem with the existing UML modeling framework described in the previous section is its failure to explicitly recognize and support the existence of two fundamental forms of classification (or instance-of flavors). Going back to the product example of Figure 2, the M_2 level really aims to address two concerns. Figure 7 shows how Video and 2001 are to be understood in terms of the UML metamodel in the OMG four-level metamodeling framework. Although regular user types (classes) at the M_1-level, such as Video, are viewed as instances of Class according to the instance-of notion explained in the previous section, user instances at the M_0-level are viewed as instances of two concepts: the instance 2001 is classified twice, first by the M_1-level (domain) type Video, and second by the M_2-level metatype Object. The rationale for this, which becomes most evident in the context of CASE tool representations, is that one would like to know that Video is the domain type for 2001 but it is also crucial to know that 2001 is an object (i.e., can have slots and links and is associated with a certain notation). If 2001 were a component instance, a CASE tool would need to render it differently and allow a different set of operations on it.

Thus, in general, it is necessary for model elements to have both a logical domain classifier (e.g., Video), defining its content and a physical classifier (e.g., Object) defining its structure and presentation. The existence of these two different forms of instance-of relationships has been pointed out by a number of researchers. Bézivin and Gerbé [2001] use the term metainstance for physical classification and instance-of for logical instance-of relationships and Geisler et al. [1998] use the terms interlevel instantiation and intralevel instantiation, respectively. In this article we use the terms logical and physical classification to distinguish the two forms.

Since the current UML framework equivocates on the true role and nature of classification, a number of peculiarities can be observed. Despite the fact that the latest version of the UML subscribes to the doctrine of strict metamodeling, in Figure 7 one can observe some obvious violations of strictness.

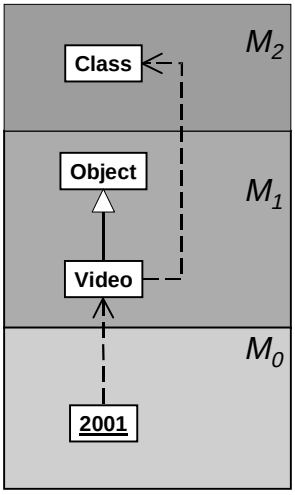

Fig. 8. Object as a prototypical concept.

(1) Object 2001 has more than one classifier (Video and Object).
(2) One instance-of relationship crosses more than one metalevel boundary.

Clearly, the desire to make statements about 2001 with regard to both its logical type (Video) and its physical structure classifier (Object) is in conflict with the idea of strictness while trying to conform to a linear metamodeling framework. The following two sections explain attempts to rectify this situation.

3.2 Prototypical Concepts

One way of tackling the need to achieve the required dual classification of model elements in a way that is consistent with the strictness doctrine is to exploit subtyping. Using subtyping one can exploit the fact that an instance of a class is also, indirectly, an instance of all superclasses of the class. Figure 8 illustrates how the dual classification of 2001 as both a Video and an Object can be achieved by making Object the supertype of Video. Note that in this scenario, the Object concept is now defined at the M_1-level (where, according to strictness, types of M_0-elements belong), and hence (indirectly) classifies all M_0-level instances as objects by virtue of being at the top of the inheritance hierarchy of all M_1-level types. The instance 2001 is thus a direct instance of Video (as before) and an indirect instance of Object (via inheritance).

The definition of "prototypical" concepts at the top of inheritance hierarchies has a long tradition in the design of class libraries accompanying popular object-oriented languages such as Java and Smalltalk. Such libraries have a class (typically called Object) at the top of the class inheritance hierarchy whose properties are inherited by all classes, and thus affect all objects instantiated from them.

Another way of looking at this design is to regard Class as the powertype of Object [Odell 1994] meaning that every instance of Class must be a subclass of Object. This ensures that all instances of classes (i.e., objects) are guaranteed

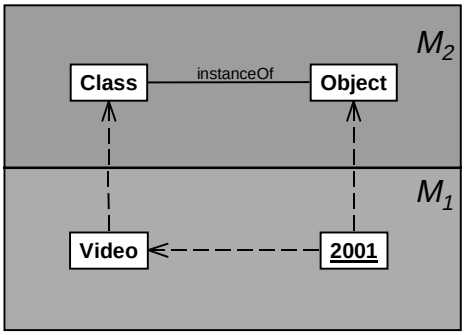

Fig. 9. UML metamodel as a language layer.

to be indirect instances of Object and thus share some standard attributes or methods. We come back to the desire to make such guarantees across two levels of instantiation (from M_2 to M_0 in this example) later in the article.

Although the prototypical concept approach achieves the desired dual classification of instances, such as 2001, it does so only with respect to logical classification. Essentially Object as used in Figure 8 may prescribe content for objects (i.e., enforce the existence of certain slots or methods such as deepCopy), but it does not really fulfill the need for defining the physical structure of objects. The fact that an object has slots and links still has to be defined elsewhere.

3.3 Metamodels as Language Definitions

The prototypical object approach mentioned above tried to fix the problem of dual classification by integrating the physical classifier (Object) into the logical inheritance hierarchy. It therefore essentially regards the logical classification dimension as being the primary one, and ignores the role of physical classification. The reverse approach is to deemphasize the logical metamodeling facet and emphasize the physical classification dimension. The role of the M_2-level is then understood solely from the perspective of language definition [Evans and Kent 1999].

The main consequence of this approach, as illustrated in Figure 9, is that logical instance-of relationships no longer play a role in determining the level structure. Although 2001 is logically an instance of Video in Figure 9, from the perspective of the metamodel both elements are viewed as just instances of language elements, on an equal footing from a metamodeling perspective. In such approaches the concept of strictness is deliberately ignored, since it is impossible to reconcile it with the notion of explicit intralevel instantiations as occurring in Figure 9 between 2001 and Video.

An advantage of this "UML as a language"[2] approach is that it offers a path to a rigorous and proven strategy for defining the semantics of model elements [Evans and Kent 1999]. If the UML metamodel is solely regarded as a language definition layer, established ways of assigning semantics to programming languages can be directly applied to the UML, enabling the exploitation

[2]M_2 is seen as a language definition level for both M_1 and M_0 levels.

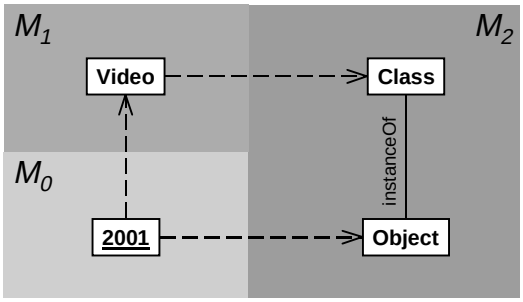

Fig. 10. Nested metamodeling architecture.

of a large body of knowledge and techniques in computer science [Harel and Rumpe 2000].

This approach also has some disadvantages, however. The first and most significant is that it provides no explicit or natural approach for meeting the needs of logical metamodeling. Apart from stereotypes and tagged values—which should really be interpreted as just a shorthand syntax for logical metamodeling—there is no way to classify types (i.e., to define types of types). Any stereotype, (e.g., <<Taxed>>) intended to achieve a logical classification of classes has to be explained as a subclass of a physical classifier making it a means of physical rather than logical classification. Emphasizing physical classification at the expense of logical classification is therefore not a truly satisfactory solution either.

3.3.1 *Nonlinear Frameworks.* A way of side-stepping this dual classification dilemma is to choose the classification flavor based on the context in which classification information is needed. The approach by Álvarez et al. [2001] abandons the idea of a linear framework and organizes the levels in a nested way (see Figure 10). The instance 2001 is then understood to have the type Video when looked at from the perspective of a domain modeler. However, from the perspective of a tool, 2001 is said to have type Object and the instance-of relationship to Video is then merely a link between two model elements. A "G" mapping is defined that can translate one of the above views into the other [Álvarez et al. 2001].

The price to pay for escaping the dilemma in this manner is that the answer to the question, "What is the type of 2001?" has now become context-dependent. The idea of a single correct answer, which transcends the perspective of the enquirer, therefore has to be abandoned. Also, whenever one needs to have a complete picture about the nature of 2001 one still needs to consult two places, Video and Object.

Another problem is that although the nested approach at least grants a level boundary between M_0 and M_1 (only from a perspective that excludes levels M_2 and higher, though) it does not really contribute to the understanding of distinct instance-of relationships. The construction of the nested hierarchy and the associated application of the "G" mapping are explained as a recursive process, which implies that the same metalevel generation principles are applied

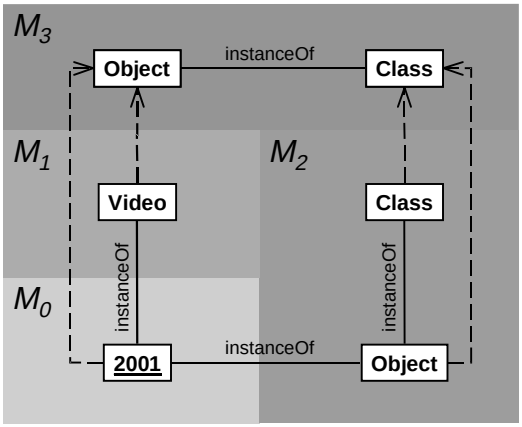

Fig. 11. Replication of concepts.

throughout. This, however, implies that the *logical* instantiation between 2001 and Video is viewed as being the same mechanism as the *physical* instantiation between 2001 and Object, clearly blurring the differences between logical and physical classification.

A problem, that has not been addressed by the work of Álvarez et al. is the "replication of concepts" problem [Atkinson and Kühne 2001b] (see Figure 11). Since a two-level "definition and use" approach is applied multiple times—once for each nesting level—concepts like Object, Class, and the instance-of relationship have to be defined time and again for each level above M_1.

In summary, the nested approach deliberately abandons the ideas of strictness and a linear framework in favor of a "more powerful" organization strategy. However, it creates new problems of its own such as not explicitly distinguishing between fundamentally different classification dimensions, making the "type-of" question context-dependent, and having to replicate the definition of concepts.

4. REARCHITECTING THE UML INFRASTRUCTURE

In the previous sections we provided a historical perspective of the motivations and influences that have helped shape the current UML modeling framework, and identified some of the major problems (i.e., dual classification, class/object duality, and replication of concepts) that have become evident, particularly in relation to the notation of strictness. We also discussed some of the research proposals for overcoming them. In the remainder of this article we draw upon these insights to propose an overhaul of the UML modeling framework that addresses the identified problems while remaining faithful to the spirit and intent of the original framework.

We describe three fundamental concepts leading to a rearchitected, coherent UML modeling framework. Although essentially independent, these build on each other to provide a significant enhancement to the existing metamodeling approach. The following section then addresses some of the ramifications of

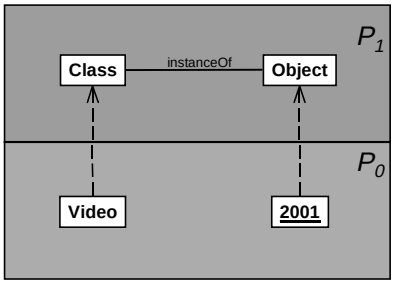

Fig. 12. Physical dimension.

these proposals on the coherence, effectiveness, and usability of the infrastructure as a whole.

4.1 Two Fundamental Metadimensions

As explained in the previous section, the underlying cause for most of the problems with the current UML metamodeling framework, especially with the application of strictness, is the failure to properly recognize and accommodate two fundamentally distinct forms of classification: logical classification and physical classification. The current UML metamodel does not explicitly distinguish between these two forms of classification and, consequently, there is no notion of how the overall shape of the modeling framework and location of model elements could be driven by these distinctions.

An important first step towards a coherent modeling framework is therefore to explicitly identify instance-of relationships as being either physical or logical. The importance of doing so has been emphasized by several researchers [Bézivin and Gerbé 2001; Geisler et al. 1998], albeit using different terminology. However, this alone does not go far enough. It is also necessary to understand what role these two forms of classification play in shaping the overall framework. In turns out that by distinguishing logical and physical classification, and mapping out distinct metadimensions for them, one can achieve a clean separation of concerns in terms of describing model elements.

4.1.1 *Physical Dimension.* The dominant classification dimension from the viewpoint of tool builders is the physical dimension. This essentially adopts the "UML as language" philosophy outlined above, and views the UML metamodel (M_2) as defining the physical classifiers (abstract syntax) from which models are constructed. Any logical instantiation that may take place within a model (e.g., between a class and an object) is essentially ignored from this perspective, and the levels are defined purely from the point of view of physical instantiation. A schematic representation of this dimension is illustrated in Figure 12. Not surprisingly it is basically the same as Figure 9 since this also illustrates the "UML as language" metaphor. However, to emphasize that we are now interpreting this as representing just physical classification, in Figure 12 we use the labels P_1 and P_0 to identify the levels.

In this dimension, the P_1 level defines (language) concepts from which user models are created. Defining 2001 as a (physical) instance-of Object defines it

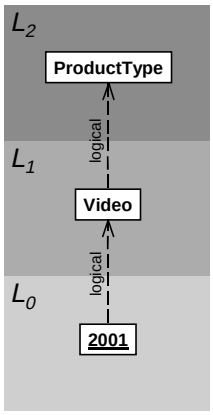

Fig. 13. Logical dimension.

as being a model element that can have slots and links, and defining Video as being a (physical) instance-of Class defines it as being a model element that can have attributes and associations. These two instance-of relationships, however, have nothing at all to say about the logical relationship between Video and 2001 and the fact that one (Video) is thought of as being a logical classifier of the other (2001). In Figure 12 level P_1 contains an association between Class and Object, which expresses the fact that objects (such as 2001) may be linked (by an instance-of dependency) to their logical domain type (Video). From the viewpoint of the physical dimension this does not constitute a level boundary, though. Creating level boundaries between logical types and instances is the role of the logical dimension.

4.1.2 *Logical Dimension.* The dominant classification dimension from the viewpoint of modelers (i.e., users of the UML) is the logical dimension. It focuses on classification within a domain, and is not concerned with the representation of concepts (i.e., with physical classification). This is illustrated in Figure 13, which shows three levels of logical instantiation. The concept 2001 is a logical instance-of the concept Video, which in turn is a logical instance-of the concept ProductType. To emphasize the fact that Figure 13 describes only logical classification we have labeled the levels L_2, L_1, and L_0. In contrast to Figure 2, we have changed the type of Video from Class to ProductType to illustrate that this dimension deals with logical classification only. The fact that Video is an instance of Class is a statement concerning its representation (its physical structure; see Figure 12), which is irrelevant in this logical dimension. Integrating the logical dimension with the physical dimension so that we fully know what Video is, is addressed in the following section.

The reason for including a third level L_2 in Figure 13 is not just to demonstrate the difference between a logical classifier (ProductType) and a physical classifier (Class). There are numerous reasons why such a logical metalevel above classes would be beneficial in the modeling and even realization of systems. Odell [1994] demonstrates the usefulness of associating information with

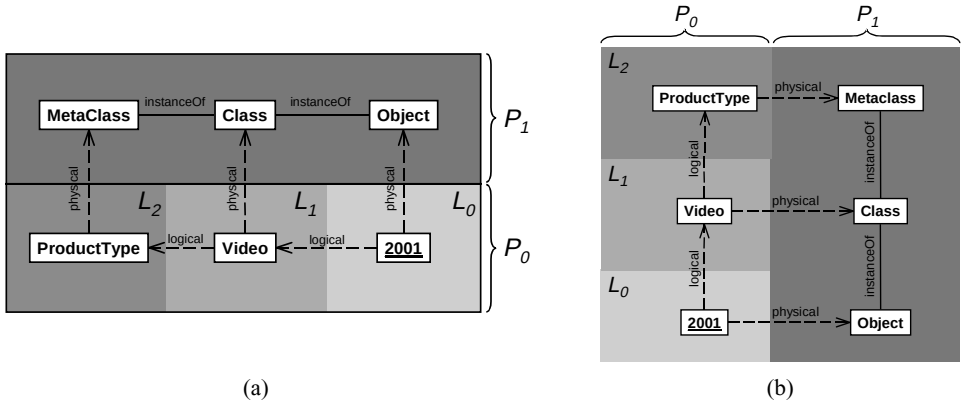

Fig. 14. Two-dimensional classification framework.

types so that they can be viewed as being instances themselves. Mili and Pachet [1998] also give a number of examples, including one showing how to express commonalities among models, which cannot be well captured with generalization. The Type Object pattern essentially describes how to emulate three metalevels with two (i.e., how to gain the flexibility of first-class types by only using classes and objects [Johnson and Woolf 1998]). Just as the application of the Visitor pattern [Gamma et al. 1994] can be regarded as a symptom of the lack of multidispatching in the language used, the application of Type Object is a symptom of the lack of logical metaclasses.

There is no a priori reason why only three levels should be used for logical classification. The principles discussed here scale up intuitively to multiple logical levels.

4.1.3 *Two-Dimensional Modeling Framework.* The key to integrating logical classification with physical classification is to view them as orthogonal dimensions. Both views are equally valid and important, and as illustrated in Figures 14(a) and (b) can be integrated into a single coherent modeling framework by associating them with the horizontal and vertical dimensions, respectively.

Figures 14(a) and (b) represent exactly the same framework, but with a different emphasis. Figure 14(a) on the left-hand side organizes the physical levels vertically and the logical levels horizontally, whereas Figure 14(b) organizes the physical levels horizontally and logical levels vertically. The difference is just one of viewpoint, corresponding to the different perspectives of tool builders and modelers as outlined above. Note that the flavor of the instance-of relationships is explicitly distinguished and that different identifiers (P and L) are used to indicate the roles of the levels (physical and logical, respectively). The collection of levels L_2 to L_0 constitutes level P_0. A similar approach, with one level containing physical classifiers "meta" to all elements in another hierarchy, is employed by Riehle et al. [2001]. However, distinguishing between physical and logical classification *within* the metamodeling hierarchy is not considered in their work.

Figure 14 illustrates another advantage for including a logical L_2 level, namely, its role as an ideal host for user stereotypes. In the depicted framework, stereotypes such as Taxed or ProductType are simply (meta-) classes at the L_2 level. Clearly the intent of these stereotypes is to partition user classes into those that are, for example, ProductTypes and those which are not. The classes themselves (such as Video) are still ordinary classes as opposed to new language elements. Therefore, the traditional way of explaining the effect of stereotype application as an extension of the M_2 (language) level with corresponding (sub-) metaclasses, really mixed up language extension (extending the set of physical classifiers) with domain type classification (extending the set of domain metatypes). The framework depicted in Figure 14 nicely separates these two concerns.

Apart from explicitly distinguishing orthogonal dimensions of classification, the main advantage of this conceptualization of the UML modeling framework is that strictness is restored as a natural and valuable discipline for metamodeling. Once the two dimensions are separated, the doctrine of strictness applies to each of them individually in a natural way. When each dimension is regarded separately, it is true to say that the instance-of relationship forms the basis for defining the level hierarchy, and thus determines the location of elements within the levels.

4.1.4 *Two or More Dimensions?.* Some may argue that two dimensions of classification are not sufficient. After all, an amphibious vehicle may be classified as a Car, a Boat, a UML object, an XMI fragment, a Java object, or an HTML page. On the surface it appears that a multitude of classification dimensions is necessary to capture all the various ways in which the amphibious vehicle instance may be classified. However, note that Car and Boat are both classifiers in the logical domain and multiple inheritance could, for example, be used to express that these different views exist on one and the same instance. In contrast, all the other classifiers refer to how to *represent* the vehicle (i.e., they are physical classifiers). In fact, they do not even classify the same instance, but each has its own vehicle representation which is related to the other instances by "correspondsTo" or "representationOf" links.

It becomes clear how fundamental the logical and physical dimensions are when one considers that to fully characterize a modeling element one always needs to say what notation was used to draw[3] it and what its domain classifier is. The latter will typically be shown using the same notation but usually by a different abstract syntax element.

4.2 Unified Modeling Elements

Explicitly distinguishing metadimensions as described in the previous section represents a big step towards a clean and coherent UML infrastructure. However, there is still room for improvement from the perspectives of both tool builders and modelers. The next stage is concerned with simplifying the P_1

[3]More precisely, what ontology was used to represent it (e.g., what its abstract syntax classifier is).

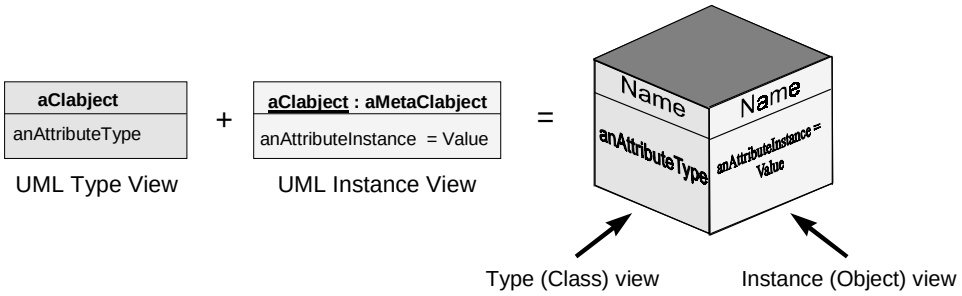

Fig. 15. Unification of class and object facets.

level to reduce complexity for tool builders while at the same time obtaining increased expressiveness for modelers.

In Figure 14, Video's physical classifier is Class. However, from the point of view of ProductType, Video is more appropriately classified as an instance; for example, it may have a tax slot[4] holding the percentage value to be used for the taxation of videos. From this viewpoint, therefore, Video should have Object as its physical classifier. Obviously, Video has both an instance facet (e.g., the tax slot) and a type facet (e.g., a price attribute).

Figure 15[5] depicts how Video—the cube at the right-hand side of the equation—can be regarded as the composition of a class and an object. This type/instance duality occurs for all model elements in an instantiation hierarchy except for those at the bottom level, which have an instance facet only.

It is possible to handle this duality by arranging for Class to inherit from Object (in the context of Figure 14, level P_1), MetaClass to inherit from Class, and so on if there are more levels.[6] This would enable both Object and Class views on Video; and Object, Class, and MetaClass views on ProductType, and so on.

However, this approach is not satisfactory for the following reasons.

(1) It introduces a superficial difference between instances of model elements at different levels and thus requires a multitude of physical classifiers.
(2) It means that the number of metalevels is dictated by the available physical classifiers and not by the needs of the modeler.

In order to see why modelers may need even more logical metalevels than we have already dealt with, suppose that 2001 itself were a only description of the movie and thus could also be regarded as a classifier of actual video copies (i.e., by representing a template for their properties). This would add an additional logical layer at the bottom of the logical levels in Figure 16, which would contain the actual copies of 2001. Although there are other ways to model the relationship between a template of a video (2001) and its copies, this is certainly a valid one as both the description of 2001 and the classifier for copies

[4]A tagged value in UML-speak, but essentially a class slot.
[5]Clabject = CLAss + oBJECT.
[6]An approach essentially followed by the Smalltalk class library.

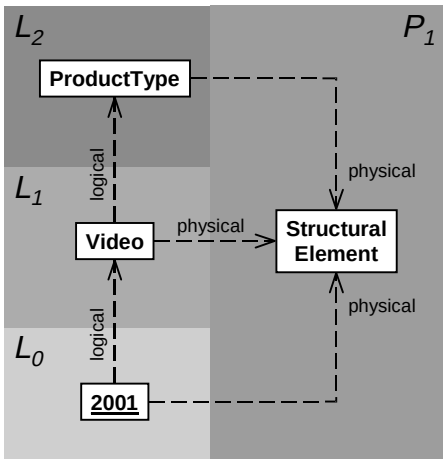

Fig. 16. Two-dimensional framework with metaclass/class/object unification.

may be regarded as two views on one modeling element. This is supported by the fact that both may be identified using the same ASIN.[7] Assuming just these four levels, we already need to add another physical classifier to P_1, an approach that obviously lacks elegance and arbitrarily limits the number of levels.

The variety of physical classifiers at P_1 is unnecessary if the instance/type duality for model elements is accepted and the bottom and top levels are treated as special cases. The essential idea is that the distinction between object, class, and metaClass is no longer needed if one StructuralElement classifier is provided. Slots, attributes, and metaattributes are then likewise unified into the notion of a field, which may contain values, types, or metatypes (model elements in general). With the exception of the bottommost level (Object), the above-mentioned views are identical anyway. Although it is true that it is not reasonable to expect a good answer to the message (e.g., numberOfInstances or setOf Subclasses from an object), this, however, cannot be expected from an abstract class (has no instances) or a final class (has no subclasses) either. Given the advantages the unification has to offer, it seems justifiable to let objects return empty sets in such cases.

The effect of treating elements as inherently having an instance and a type facet on the two-dimensional framework described above is illustrated in Figure 16.

Associating model elements with their correct levels was trivial with the "one physical classifier per L level" approach since objects always reside at the L_0 level, classes always at the L_1 level, and so on. To do this in the unified version one simply equips StructuralElement with a level attribute so that model elements in the L levels have values 0, 1, and so on, corresponding to the level at which they reside. The value of this level slot would also be used to choose an appropriate rendering of the model element and to enable certain operations in a tool, and the like.

[7]Amazon Standard Identification Number.

4.2.1 *Redundant Physical Classification.* The previous section proposed to unify Class and Object into StructuralElement and do the same for other type/instance classifier pairs in the UML metamodel, for example, Component and ComponentInstance. By doing this we removed the differences between a type and its instance but maintained the distinction between several modeling element types with both an instance and a type facet. The superstructure within P1 would still distinguish among structural elements, behavioral elements, deployment elements, and so on. The unification idea can be taken further, however, by unifying all these domains as well. All that is essentially needed in this case is a ModelElement concept. With such a shrunken-down version of the superstructure in place, this advanced unification would—next to simplifying the P_1 level and enabling an unbounded number of logical metalevels—have another very important effect. Since all logical model elements are now instances of the same physical classifier (ModelElement), the physical instance-of relationships have become essentially redundant. For a modeling tool it is no longer necessary to maintain the link to a physical classifier (such as Object or ComponentInstance), because every logical model element is, by definition, an instance of the physical classifier ModelElement. Obviously, another way then has to be found to distinguish an object from a component instance. Section 5.1 explains how this can be achieved in a way that makes the UML superstructure adaptable for modelers in a natural fashion.

4.3 Deep Instantiation

The unification described above elevates model elements to proper citizens in a multilevel metamodeling hierarchy but the instantiation mechanism has not yet been upgraded accordingly. Traditional instantiation is sufficient when dealing with only two levels (e.g., classes and objects) but should ideally be enhanced for a multilevel instantiation hierarchy. The problem with traditional instantiation is that a type may only specify properties of its direct instances but has no bearing on, for example, the instances of its instances. This is why we refer to it as shallow instantiation.

An example of when it is necessary to make statements about instances across two levels is to require that all products (i.e., instances of some product type such as Video or DVD) have a price slot. The natural place for making this statement is at ProductType, since every instance of ProductType is required to have a price attribute. Establishing this is possible either by associating a constraint with ProductType, which requires ProductType instances to have a price attribute, or by using the powertype concept (see Figure 17). The purpose of specifying ProductType as the powertype of Product is to enforce the requirement that every instance of ProductType (e.g., Video) inherits from Product and thus acquires a price attribute by inheritance. The supertype Product is, hence, used to specify what is expected from all instances of ProductType instances. In this example, one ensures that any product, such as 2001, will have a price slot no matter what product type (e.g., Video, DVD, CD) it is.

We argue that neither constraints nor the powertype concept appropriately addresses the need to specify properties across more than one level of

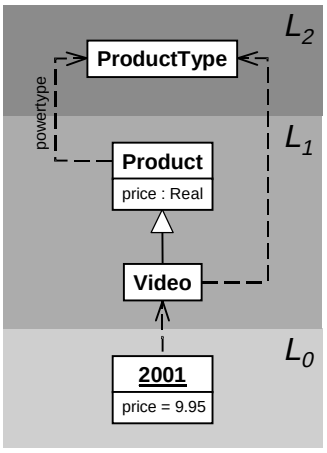

Fig. 17. ProductType as a powertype.

instantiation. The next section elaborates on how to achieve this more naturally by enhancing instantiation to be "deep" rather than "shallow" and motivates why the scope of a property can sometimes even extend over more than two levels.

4.3.1 *Potency.* Deep instantiation is a conservative extension of the traditional two-level, shallow instantiation mechanism. It subsumes shallow instantiation but enhances it to allow information to be carried over more than one instantiation step. The key idea is to assign a potency value to model elements and their fields, indicating how many times they can be instantiated. Instantiating a model element therefore amounts to reducing its level by one and also reducing its potency and the potencies of all its fields by one. Traditional classes, attributes, and associations have potencies of 1 since they can be instantiated exactly once, and traditional objects, slots, and links have potency 0 since they cannot be instantiated anymore. Instantiating a potency 1 element such as Video creates a potency 0 element, as required. Figure 18 illustrates how the intention of the modeling scenario of Figure 17 can be expressed with deep instantiation.

There is no need for a Product concept anymore, since the price attribute in Video is enforced as an instance of the potency 2 price (meta-)attribute in ProductType. Instantiating ProductType to Video turns its potency 2 price field into a potency 1 field, which corresponds to a regular attribute as desired. Although types traditionally may only describe the instance facet of elements, potency values higher than one, therefore, can be used to also describe the type facet (and metatype facet, etc.) of instances.

Note how well the potency concept matches the conceptually unbounded number of logical metalevels we obtained from unifying the physical classifiers. With the appropriate potency value a property can be enforced across any number of instantiation steps. For instance, if we again assume that 2001 is a classifier for actual copies of 2001 rather than a bottom-level instance, we can

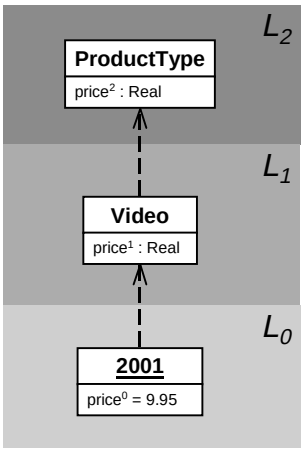

Fig. 18. Deep instantiation.

model price as a potency 3 field in ProductType so that only copies are associated with prices. In contrast, copy classifiers (e.g., 2001) only specify that copies have a price, which may vary depending on the place of manufacture of the copy.

When a modeler only uses two levels (classes and objects) deep instantiation—using appropriate notation conventions—looks and feels exactly like traditional shallow instantiation. Only if a modeler chooses to use levels above L_1, do the concept and associated notation for deep instantiation come into play, providing enhanced expressiveness. This added expressiveness exactly matches the increased modeling potential enabled through additional logical metalevels.

Together the level and potency properties of modeling elements allow very concise and precise definitions of the properties of model elements with respect to instantiation, and are capable of obviating the use of constraints, powertypes, and prototypical metaclasses on many occasions [Atkinson and Kühne 2001b].

5. TOWARDS A COMPLETE UML INFRASTRUCTURE

In the previous section we described three complementary enhancements to the current UML modeling framework which would place it on a sound footing and enhance the capabilities available to modelers. To avoid overcomplicating the description of these ideas, we presented them in as concise and focused way as possible. However, if the rearchitected framework is to serve as the infrastructure[8] for forthcoming versions of the UML some important issues remain to be addressed. Some ideas for addressing these issues are discussed in this final section.

[8]Conforming to the UML 2.0 RFP we use the term infrastructure to refer to the combination of metamodeling hierarchy and other related aspects, such as semantics.

5.1 Language Elements versus Library Elements

One of the major objectives for the new UML infrastructure is to enhance the extensibility of the UML, so that it is more tailorable to user-specific domains and the vision of a family of languages can be fully realized. Since the UML is essentially an object-oriented language, it is valuable to draw parallels to the way in which the issue of customization is addressed in object-oriented programming languages.

Among the key concepts that distinguish object-oriented languages from previous languages is that regular users can add their own types (with associated behavior) to the predefined set of types available for creating new programs. In the days of FORTRAN, properly integrating the concept of a point type with polar coordinate representation would have required a language extension. Today's object-oriented languages allow the definition of classes that become part of a library for use in the creation of future programs and therefore are much more tailorable to suit the needs of a user in a certain specific domain. The concepts available to users of an object-oriented programming language are therefore defined in two basic ways:

—core language concepts, and
—library classes.

Moreover, in well-designed languages the syntax is usually set up so that it is impossible to tell (from the syntax alone) whether a core language type or a (user-defined) library class is being used. Thus a strategic issue for the design of an object-oriented language is what concepts to define within the language core and what concepts to define in the predefined library of classes. The strategy adopted by most object-oriented languages is to make the core language definition as small as possible and to define as many concepts as possible within the class libraries. This helps to

—keep the language definition small and simple,
—keep the virtual machines stable even in the event of considerable (library) concept restructurings, and
—give users maximum tailorability.

Users may change or extend a library but cannot change the core language concepts (e.g., object identity, instantiation, message sending, etc.).

The distinction between language elements and library elements has never been adequately addressed in the previous definitions of the UML, partly because of the lack of clarity about the distinct forms of classification. With logical and physical classification separated into distinct separate dimensions the choice becomes much clearer. Core language concepts appear within the top physical level (P_1 in Figure 14) whereas concepts defined as part of the predefined library (i.e., modeling standard) appear in the top logical level (L_2 in Figure 14). Thus, following the lesson of object-oriented programming languages, minimizing the core language definition would correspond to minimizing the size of P_1 and placing as many concepts as possible in L_t, where L_t is the top logical level. In other words, the lesson learned from object-oriented

languages implies that physical classification should be minimized and as many concepts as possible should be placed within the logical classification hierarchy. A consequence of this approach is—when used in combination with a potentially unbounded number of logical levels—that the logical hierarchy would grow downwards, that is, growing new bottom levels instead of growing new top levels which should be accounted for by an appropriate numbering scheme.

The library idea is applicable to many concepts in the current UML metamodel which might at first appear to be natural parts of the language definition. In fact, most of the UML's superstructure could be moved to the logical level in order to achieve the separation of concerns exemplified by object-oriented languages and their libraries. In order to achieve the same benefits of tailorability, the only concepts or mechanisms that the UML core needs to address are those related to

1. type/instance instantiation,
2. specialization,
3. graph-based models,
4. presentation (concrete syntax), and
5. reflection.

The first two items basically contribute the essence of object-orientation. The next two allow the arrangement of concepts in the form of graphs with a graphical notation (the hallmark of the UML). The last item provides standardized access to model information (see Section 5.3).

An approach where each level has to reestablish the above core concepts in addition to the logical concepts (i.e., using physical classification only for building levels) directly leads to the "replication of concepts" problem noted before. Such an approach actually made sense if the fundamental underlying paradigm can change from level to level and is thus needed in the most general case. However, when object-oriented modeling is accepted as a general theme across all levels then it is better to capture its core features once (at level P_1) and therefore be released from re-creating object-oriented modeling facilities for each (physical) level time and again.

An interesting property of this approach is that most of the current UML superstructure would not reside at a language definition level (i.e., M_2) anymore, but be provided as a modeling library at the top logical level, L_t. As such it would be extendable by users and enable tailorability without involving language extensions and the associated consequences of dealing with language standardization, language semantics, tools relying on language invariants, and the like. The "Unified Modeling Library" (UML) would be the core profile to be extended by all custom profiles, and by being based on logical classification such extensions would not have to mix a language extension metaphor with a library extension metaphor as is currently the case.

5.2 Semantics

One of the most important issues to be addressed (which has actually driven much of the research work in this area) is the provision of precise and

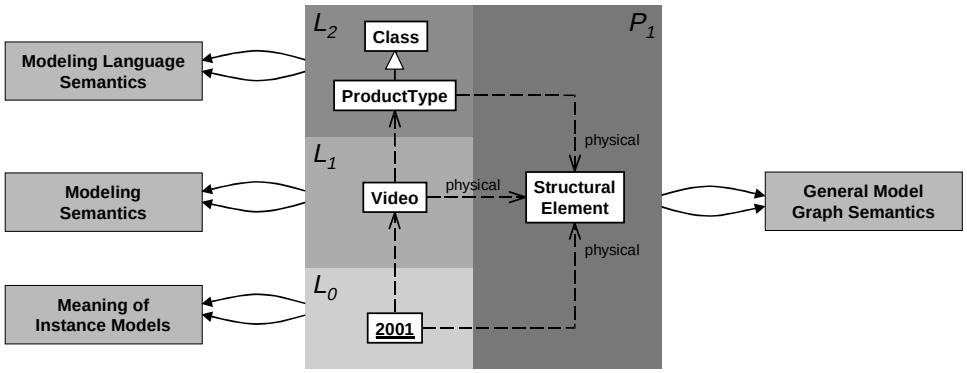

Fig. 19. Assigning semantics.

unambiguous semantics for UML models. At present, the semantics of the UML are primarily described in natural language, but this approach has well-known limitations [Harel and Rumpe 2000]. If the UML is to receive adequate and consistent tool support, the definition of semantics has to be placed on a more formal footing.

The important point to bear in mind, when addressing this question, is that different approaches can be used for the core and the library parts, as discussed above. For the core language semantics in P_1, the most natural approach is to exploit the tried and tested techniques used for programming languages, and define the semantics of language concepts in terms of a mapping to a semantic domain [Evans and Kent 1999; Harel and Rumpe 2000]. However, for the library elements defined in the logical levels a different approach can be used. It is likely that OCL statements will play an important role in defining the well-formedness rules related to the use of specific model elements (based on their meaning) but the actual definition of their meaning could apply some domain-specific formal approach, or in the worst case could still use natural language.

5.2.1 *Notation.* Defining the presentation of model elements (i.e., their visual appearance) is not considered to be as problematic as finding a proper semantics, although the complications of a visual notation in comparison to a textual one are recognized. One formal approach is to regard the model elements as abstract syntax and their notation as the associated concrete syntax [Evans and Kent 1999].

It has to be noted, though, that concrete syntax in the UML has to be flexible in several ways. One of the capabilities of the UML is to allow different users to use different concrete syntax (i.e., icons) for the same basic concept. Among the intentions of introducing the stereotype mechanism, for example, was allowing users to associate new icons with modeling concepts. Also, different views, such as, an analysis view versus a design view, may use differing levels of refinement with regard to the visual presentation of the underlying model. Finally, users must be able to define new concrete syntax for any new specialized concepts that they introduce.

This highlights the fact that the UML needs a dynamic concrete syntax. Dynamic syntax means that the way in which a model element is rendered (i.e., presented), depends on more than just its physical type and that a CASE tool should consider additional data items (e.g., the state of the element, the current view on it, etc.). In the current version of the UML, such capabilities are partially addressed with the notion of presentation elements. In short, the issue of concrete syntax in the UML should be understood as being dynamic in nature and being related to the model elements used to capture presentation.

Note how the unification of physical classifiers (see Section 4.2) made the presentation of elements dependent on the state (i.e., the level value) of a model element rather than the type. From the perspective of the physical P_1 level, the logical classifier (e.g., a metaclass representing a stereotype with an associated notation) is just another piece of information—encoded into the state of the model element—as to how to render the model element.

The issue of syntax also has a relationship to the possible role of stereotypes in the new UML infrastructure. With an improved understanding of the role of logical metamodeling, such as that offered in the previous section, stereotypes can best be understood as a shorthand way of applying logical metamodeling combined with control over dynamic presentation. Hence the required notion of presentation elements could actually fulfill a much larger role than just generating a visual notation. They could be used to generate other forms of representation, as outlined in the following section.

5.3 The Role of the MOF

Although the term MOF has its origins in the OMG's Object Management Architecture as a repository for dynamically accessible metainformation and for supporting model interchange in the first four versions of the UML, it quickly became associated with (i.e., treated as being the same as) the meta-metamodel at level M_3. Unfortunately, these two interpretations are not compatible in the original linear metamodeling framework. If the MOF is identified with the M_3 meta-metamodel then it can neither act as a reflection interface nor provide model interchange support for *all* the levels below (M_2 to M_0), unless one abandons the concept of a strictly *linear* modeling framework.

The most natural interpretation of the name MOF, from a purely terminological perspective, would be to return to the original intent of supporting reflection and model interchange. These responsibilities correspond to items 5 and even 4 in the list of capabilities to be provided by the core (P_1) language. In the original one-dimensional, linear modeling framework, this would have required the MOF to manifest itself at every level (except the bottommost), since it needs to provide interfaces (an API for tools to gain metainformation) to elements at all levels.

As illustrated in Figure 20, this is not necessary in the two-dimensional framework. Here, the MOF is simply regarded as representing a common and standardized interface to the physical structure of model elements, which is exactly the view required by tools. This design makes sense, especially if one adopts the "UML as a library" approach since then even the UML's

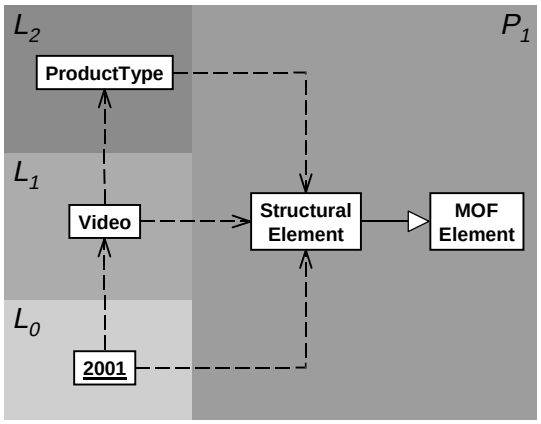

Fig. 20. The MOF as a level spanning reflection interface.

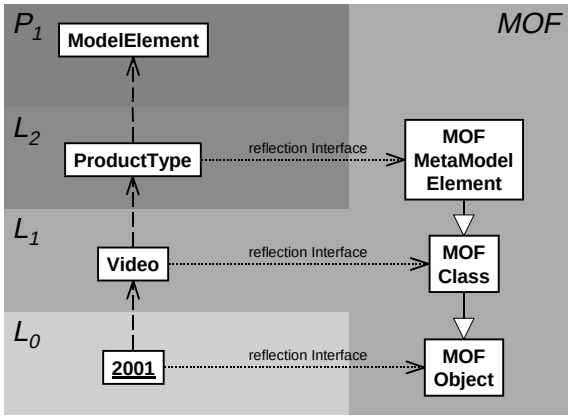

Fig. 21. MOF as another modeling space.

superstructure is accessible through the MOF interfaces. In this design one may even go so far as to refer to the whole of P_1 as the MOF, thereby assigning the responsibilities of a meta-metamodel, reflection interface, and model interchange support to the MOF again.

An alternative design, which is more appropriate if one wants to retain the concept of a physical UML language level is to regard the MOF as something external to the UML metamodeling framework, which can see the latter as one entity (i.e., disregard level boundaries). In the terminology of Atkinson and Kühne [2001a] one would regard both the UML metamodeling framework and the MOF as "modeling spaces," which are not forced to fit into a single-level hierarchy but can have their individual hierarchies allowing intermodel space relationships that do not have to respect level boundaries. See Figure 21.

In the previous section we hinted at a relationship between the concept of concrete syntax and the MOF's role to support model data interchange. The key to recognizing this connection is to generalize the requirement to support

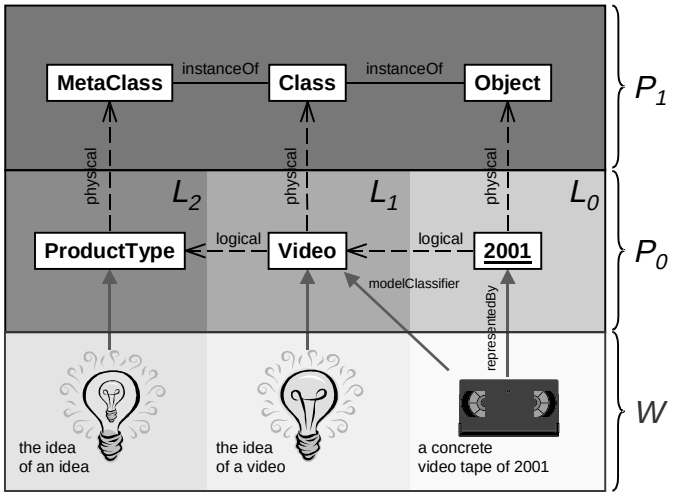

Fig. 22. Modeling and the real world.

multiple visual *presentations* of model elements to the support of multiple *representations* of model elements. An XMI representation of a model element (meant for interchange) is a rendering of the underlying model element, just like the visual representation of it on a drawing canvas. Ergo, the facilities dealing with model element presentation (discussed in connection with concrete syntax above) could be viewed as a part of the general MOF-related need to support multiple representation of model elements.

5.4 The Location of Real World Objects

Another important point to be addressed is the question of how the two-dimensional modeling framework relates to the real world. The question arises since especially in data modeling contexts, it is plausible to think of the actual physical data (e.g., a record in a database or an XMI fragment) as being an instance of a modeling element type (such as Video). In more analysis-oriented applications of the UML, even real people, for example, a human being called "mike" could be regarded as being instances of a type Person or a role Actor.

The question, therefore, arises as to what the relationship of L_0 in the two-dimensional framework to the above interpretation of M_0 is. Not surprisingly, L_0 contains instances that are *logical* instances of user types at the L_1 level. Consequently, L_0 hosts instance models whose elements *represent* things in the real world (see Figure 22). We present Figure 22 with physical classification aligned to the vertical dimension in order to show the one-to-one correspondence of levels P_1, P_0, and W (real world) to the original M_2, M_1, and M_0 levels.[9]

On the top we have a language definition level, which may or may not use the unification of physical classifiers as presented earlier. Below we have a modeling level (P_0) containing all modeling artifacts whether they relate to instance models (L_0), conceptual models (L_1), or domain metamodels (L_2). As

[9]From a modeler's perspective Figure 22 should be turned 90 degrees to the right.

was the case with the M_1 level, we can think of the instance-of relationship between 2001 and Video as an intralevel instantiation. The two-dimensional framework places both instance-of flavors (logical and physical) on an equal footing, though, and thus designates logical level boundaries within $P_0(M_1)$. Moreover, the two-dimensional framework separates the responsibilities of the original M_2 level into the P_1 (language definition) and L_2 (user-defined domain metatypes, e.g., stereotypes) levels, respectively. Finally, at the bottom level we have real-world things, which first and foremost are in a "represented-by" relationship with modeling elements. The 2001 instance stands for (i.e., represents) a thing in the real world. As an element of an instance model it could be used in simulations when the modeler is trying to validate some assumptions about actual configurations. It could also directly be used as a prototypical specification for real world things, in order to see whether, for example, some actual tape conforms to the information contained in the 2001 instance.

Such conformance checking, however, is usually not accomplished with prototypical instances from instance models, but by using a type (e.g., Video) from the conceptual model (at L_1). This is depicted by the "modelClassifier" relationship between the real tape and the conceptual model element Video. In the original one-dimensional framework one would expect the real type directly below, that is, vertically aligned with, Video and a general instance-of relationship between them. Note, however, how the two-dimensional framework explicitly distinguishes between physical classification (e.g., between Class and Video) and model classification (e.g., between Video and the real tape). We chose not to do the vertical alignment of the real tape with Video in order to emphasize the role instance models can play and to illustrate the correspondence between model elements and things in the real world. For instance, Video is not only a model classifier for real tapes, it is also a representative of the real-world idea of a video. It represents the mental concept that defines the intension of the Video concept, that is, the description of what qualifies as a videotape and what does not.

5.5 The Shape of the Infrastructure

The final question we wish to address is how the architecture of the UML infrastructure should be conceptualized. As explained, a great step forward in clarity and consistency is achieved if logical and physical classifications are distinguished and regarded as separate dimensions. Both subfigures of Figure 14 visualize the separate dimensions. Note that the core language constructs (at P_1) can be regarded as being "meta" to all the logical metalevels. Adopting the idea—which is also embodied in the nested hierarchy approach of Section 3.3.1—that one level can see multiple levels at once, ignoring any intralevel instantiation of the target level, is actually unavoidable with a shallow instantiation scheme. If P_1 were a linear extension to the L-level hierarchy then whatever is specified in, for example, Object, could only have an effect on L_2-level elements. Using shallow instantiation, the instance facet of an L_2-level instance (created from a P_1-level element) could not further be used as the type facet for L_1-level elements (still assuming the situation with P_1 on top of L_2).

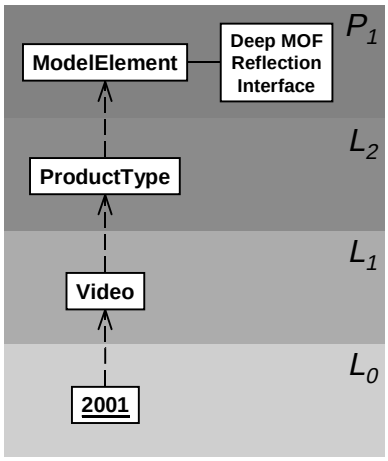

Fig. 23. Aligning logical and physical classification.

In fact, this points out that the only sound interpretation of the original linear metamodeling framework is one that assumes intralevel instantiation within M_1 (i.e., locates modeled instances at the M_1 level as well). Otherwise, the effect of M_2-level elements, such as Object, on M_0 level elements cannot be explained within a strictly linear framework.

Interestingly, though, adding the unification of physical classifiers and deep instantiation to the picture removes the need for the first proposal, the distinction between logical and physical classification. Since deep instantiation enables information and properties to be carried over multiple logical levels, it is no longer necessary to define everything relative to a level immediately next to that level (i.e., maintain a two-dimensional framework). Deep instantiation allows information pertaining to a level to be defined in any higher level. As a result, the physical level P_1 can actually be put on top of L_2 without losing any means of descriptive power (see Figure 23). The type facet of ModelElement (and associated MOF interfaces) would be replicated at every logical level (except L_0) using the notion of dual fields [Atkinson and Kühne 2001b].

It is, hence, possible to view the core language definition (P_1) as representing the topmost level in the level hierarchy. As illustrated in Figure 23, this creates a four-level architecture with a level at the top, which corresponds to the intent of the original M_3 meta-metamodel. Note, however, that interpreting this as resembling the original four-level architecture (M_3 to M_0) has to be done with caution. The architecture shown in Figure 23 realigns logical and physical classification to the same dimension. It should suit those who have interpreted M_0 elements as being modeled instances (instead of real-world things), that is, who assumed logical classification between M_1 and M_0 elements. The architecture is unusual to those who assumed intralevel instantiation between classes and objects at M_1 and viewed physical classification as the only instance-of relationship, which gives rise to metalevels. In particular, real-world things would need to be shown sideways to the hierarchy depicted in Figure 23.

Whether it is better to use two dimensions for physical and logical classification, respectively, or whether deep instantiation should be used to realign the two forms of instantiation into one again, is currently an open question.

6. CONCLUSION

In this article we have traced the history of interpretations of the UML modeling framework, and have identified some fundamental problems in the way it is currently organized. We then distilled recent research work into three basic proposals for rearchitecting the modeling framework to meet the requirements for a new UML infrastructure, in particular, addressing tailorability for modelers in specialized domains. Finally, we discussed some further issues related to the interpretation and design of the new architecture.

The first proposal was to base the architecture on an explicit separation of two basic forms of classification (instance-of relationships): physical classification, which classifies model elements according to their language classifiers, and logical classification, which classifies model elements according to their logical role in the modeled domain. This separation has a number of important benefits. In particular, it

—clarifies the distinction between language-oriented metamodeling and domain-oriented metamodeling, and thus opens the way for a sound and optimal balance between the definition of the UML in terms of language features and model library content;

—provides a natural approach for problem-oriented (domain) metamodeling and allows an arbitrary number of logical metalevels as befits the modeling problem in hand;

—provides a sound and natural interpretation of stereotypes and tagged values in terms of logical metamodeling. The notation for stereotypes can be viewed as a shorthand way of performing logical metamodeling; and

—restores the doctrine of strictness as a natural way of constructing metalevels according to instance-of relationships in each of the metadimensions (logical and physical).

Although some may argue that restoring the usefulness of strictness is of questionable value, it has to be noted that the strictness discipline was instrumental in uncovering and understanding the subtle problems of the original presentation of the metamodeling framework. In the presented two-dimensional framework, the strictness discipline is fully applicable and—like any guideline—provides help in staying away from unclear scenarios.

The second proposal integrated the instance and type facets of model elements to provide a single unified view of all elements within the modeling framework. This has the following important benefits. It

—removes the artificial distinction between instances, types, metatypes, and the like, and by providing a single unified concept enables unbounded logical metalevels without implying changes to the superstructure whenever another level is needed;

—allows relationships between model elements representing instances and types to be controlled by simple and explicit constraints; and
—opens up the possibility of an alternative organization of the infrastructure, consistent with the explicit separation of logical and physical classification.

The third proposal was to enhance the semantics of (logical) instantiation to recognize the unified view of model elements and support the transfer of information across multiple instantiation steps. This deep instantiation mechanism has the following benefits. It

—allows the natural modeling of real-world scenarios where information transcends more than two model levels;
—allows commonly occurring constraints between the type and instance facets of model elements to be expressed in a concise and natural way; and
—reinforces the possibility of an alternative organization of the infrastructures, which realigns the logical with the physical classification dimension.

It is important to appreciate that although the proposals reinforce one another, they are essentially independent and can be adopted independently. In combination they enable the UML's superstructure to be viewed as part of a modeling library, fully empowering modelers to adapt it to their needs. However, just adopting the type/instance facet unification, perhaps combined with deep instantiation, would also be of significant value to modelers, without implying such a radical paradigm shift for tool builders.

Of the three proposals, the first is probably the most important, since it alone enables the metamodeling concepts used in the UML infrastructure to be placed on a sound footing, and the concept of strictness to be restored as a way of using these concepts in a disciplined way. A nice feature of this proposal is that it offers a clearer understanding of how the concepts in the existing architecture can best be evolved, and how the variety of research-based proposals fits into the picture.

The sharper distinction between the core UML language (physical) and the predefined library (logical) opens up distinct strategies for defining the meaning of UML features. The semantics of the core language is probably best described by language-oriented approaches, whereas the semantics of library features may best be described by alternative domain-specific approaches. Both approaches are mutually compatible.

The approach of viewing diagrammatic (i.e., visual) representations of a model as merely a special case of the multiple possible representations (e.g., XMI-based, database-based) opens up the possibility of a clearer role for the MOF as a part of the infrastructure responsible for model representation and reflection.

Finally, the enhanced understanding of the instance-of relationships within the modeling framework, particularly the distinction between the logical and physical dimensions, also offers a more natural understanding of how the model elements relate to the real-world subjects of the model.

As we have clearly stated, some of the discussion points addressed in the last section of this article have not yet been fully resolved. Nevertheless we believe

that the three basic proposals we have presented lead the way to a sound footing on which a tailorable UML, supporting a family of languages, can be placed.

ACKNOWLEDGMENTS

The authors would like to thank Jean Bézivin, Shridar Iyenghar, Stuart Kent, Cris Kobryn, Michael Latta, Paul Sammut, Bran Selic, Brian Henderson-Sellers, and Friedrich Steimann for many stimulating discussions of the ideas presented in this article.

REFERENCES

ÁLVAREZ, J., EVANS, A., AND SAMMUT, P. 2001. Mapping between levels in the metamodel architecture. In *Proceedings of the Fourth International Conference on the Unified Modeling Language*, M. Gogolla and C. Kobryn, Eds., Lecture Notes in Computer Science, vol. 2185, 34–46.

ATKINSON, C. AND KÜHNE, T. 2000. Strict profiles: Why and how. In *Proceedings of the Third International Conference on the Unified Modeling Language*, Lecture Notes in Computer Science, vol. 1939, 309–322.

ATKINSON, C. AND KÜHNE, T. 2001a. Processes and products in a multi-level metamodeling architecture. *Int. J. Softw. Eng. Knowl. Eng. 11*, 6, 761–783.

ATKINSON, C. AND KÜHNE, T. 2001b. The essence of multi-level metamodeling. In *Proceedings of the Fourth International Conference on the Unified Modeling Language*, M. Gogolla, C. Kobryn, Eds., Lecture Notes in Computer Science, vol. 2185, 19–33.

BÉZIVIN, J., AND GERBÉ, O. 2001. Towards a precise definition of the OMG/MDA framework. In *Proceedings of Automated Software Engineering*, ASE'2001 (San Diego, November).

CDIF TECHNICAL COMMITTEE. 1994. CDIF Framework for modeling and extensibility, Electronic Industries Association, EIA/IS-107, January.

EVANS, A. S., AND KENT, S. 1999. Meta-modeling semantics of UML: The pUML approach. In *Proceedings of the Second International Conference on the Unified Modeling Language*, B. Rumpe and R. B. France, Eds., Lecture Notes in Computer Science, vol. 1723.

GAMMA, E., HELM, R., JOHNSON, R. E., AND VLISSIDES, J. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass.

GEISLER, R., KLAR, M., AND PONS, C. 1998. Dimensions and dichotomy in metamodeling. In *Proceedings of the Third BCS-FACS Northern Formal Methods Workshop* (September). Springer-Verlag, New York.

HAREL, D. AND RUMPE, B. 2000. Modeling languages: 2000. Syntax, semantics and all that stuff—Part I: The basic stuff, Tech. Rep. *MCS00-16*, Faculty of Mathematics and Computer Science, The Weizmann Institute of Science, Israel.

HENDERSON-SELLERS, B. 2002. The use of subtypes and stereotypes in the UML model. *J. Database Manage. 13*, 2, 43–50.

HENDERSON-SELLERS, B. AND BULTHUIS, A. 1997. *Object-Oriented Metamethods*. Springer Verlag, New York.

JOHNSON, R. AND WOOLF, B. 1998. Type object. In *Pattern Languages of Program Design 3*, Robert C. Martin, D. Riehle, and F. Buschmann, Eds., Addison-Wesley, Reading, Mass.

KOBRYN, C. 2001. UML 2001: A standardization odyssey. *Commun. ACM 42*, 10, 29–37.

MILI, H. AND PACHET, F. 1998. Patterns for metamodeling. Available at http://citeseer.nj.nec.com/44259.html.

ODELL, J. 1994. Power types. *J. Object-Orient Program.* (May).

OMG. 2001. OMG Unified Modeling Language specification, version 1.4. OMG document ad/00-11-01.

RIEHLE, D., FRALEIGH S., BUCKA-LASSEN, D., AND OMOROGBE, N. 2001. The architecture of a UML virtual machine. In *Proceedings of OOPSLA'01*. ACM Press, New York, 327–341.

Received November 2001; revised July 2002; accepted October 2002