Coherent Support for Models at Run-Time through Orthogonal Classification

Colin Atkinson and Matthias Gutheil

Chair of Software Engineering, University of Mannheim
A5, 6, 68131 Mannheim, Germany
{atkinson, gutheil}@informatik.uni-mannheim.de

Abstract. In order to allow applications to gain the maximum advantage from models at run-time it is necessary to store and provide access to model information that spans multiple ontological (logical) levels, including objects that represent instances of typical model classes. This is best achieved by means of an orthogonal classification architecture (OCA) which regards all model elements as having two distinct types - a linguistic type and an ontological (logical) type. In this position paper, we explain the difference between the OCA and the traditional UML infrastructure, and outline some of the main concepts involved in its realization. In particular, we introduce the notions of clabjects, fields and potency and explain how to achieve a uniform graphical representation of connectors.

Keywords: Meta-Modeling, Multi-Level-Modeling, Clabjects.

1 Introduction

The status of "instances" of classes has been the subject of debate since the early days of the UML. In the first few versions of the UML (1.0 … 1.x [5]) instances were portrayed as occupying the M_0 level of the four-level OMG modeling infrastructure and no explicit distinction was made between "real world" instances and their representation within a model. In more recent versions of the UML (2.0 ... 2.x [6]), however, an explicit distinction between the two was introduced. The former are now described as occupying the M_0 and the latter (referred to as *instance specifications*) are portrayed as occupying the M_1 level, alongside their classes.

The status of "instances" is important because they are one of the main ways in which the "meaning" or effect of models is manifest at run time. In traditional software engineering projects, where the UML is viewed as a vehicle for analysis and design, the status of instances of UML classes is not a major issue because their creation and management is delegated to implementation technologies such as database management systems or programming language run-time systems. In the UML, instances have traditionally only been used to provide "illustrative" snapshots of possible system states rather than to provide a "real" record of the current state of a system or a real world scenario.

As long as class instances are only used in an "illustrative" way there is no real need to worry about fitting them into an infrastructure in such a way that they are readily accessible and manipulable at run-time. However, when the goal is to use UML model data to mirror or represent the run-time state of a system - for example, as a substitute for a database or a program's run-time state - instances of classes need to be treated as "first class" citizens with the same status as class and metaclasses etc. Although the current generation of model repositories based on MOF and EMF provides flexible run-time access to classes and metaclasses, their support for instances is limited and has been hampered by the "second class" status historically given to objects and object diagrams in the UML. The introduction of instance specifications in UML 2.0 was a step in the right direction, but still left the fundamental problems with the UML infrastructure only "half addressed". In particular, it did not address the fact that model elements in the M_1 have no direct relationship to the MOF although it is meant to be the "format" in which they are stored, and it leaves the precise status of logical metaclasses (i.e. logical classes of classes) unresolved.

In [1] Atkinson and Kühne explain how many of these problems can be avoided by modeling information in an orthogonal classification architecture (OCA) that unifies the treatment of model data at all logical levels. The original proposal was put forward as an alternative to the traditional UML infrastructure, and thus focused on development-time modeling rather than the manipulation of models at run-time. However, since it places instance information (i.e. resources in ontology terminology [3]) on an equal footing to classes and other model data, the OCA turns out to be an ideal foundation for supporting "models at run time". In this position paper we provide an overview of the principles behind the OCA and describe the key features of our prototype realization.

2 Basic Principles of OCA

The basic structure of the OCA is illustrated in Figure 1. The basic idea is to organize model information in terms of two distinct (i.e. orthogonal) forms of type (i.e. instance of) relationships – linguistic types, which characterize the basic way in which model elements are represented (i.e. their information representation format) and logical, or ontological types which characterize what model elements represent (i.e. their logical content). As can be seen in Figure 1 these two forms of typing are completely orthogonal to one another and form their own level hierarchies according to the principles of strict modeling [1].

The levels L_2, L_1 and L_0 are the so called linguistic levels and characterize the linguistic typing hierarchy in a similar way to the current UML infrastructure. The bottom level is regarded as being the "real world" or the real state of the system that is the subject of the model. The middle level, L_1, contains all "user" model elements organized into different ontological (or logical) levels O_1, O_2 and O_3. Like the UML infrastructure, logical class instances (e.g. Einstein) occupy the same level as their types (e.g. Professor). However, in contrast with the UML infrastructure there can be an arbitrary number of logical typing levels. In Figure 1 there are only three, but in

general there can be more depending on the application. Note that we number the levels in the opposite way to the usual UML convention to accommodate the fact that the number of ontological level is variable. The base level (the most "meta") level is assigned the level zero, the next level containing instances of level zero elements, is assigned the level one and so on.

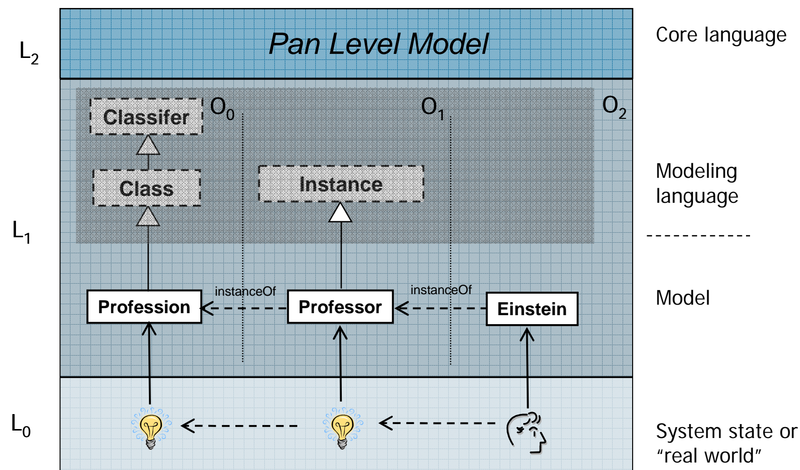

Figure 1 Orthogonal Classification Architecture

An important difference between the OCA as depicted in Figure 1 and the UML infrastructure is that there are only three rather than four linguistic levels. The "language definition" role played by standard "meta "models in the current UML infrastructure is instead played by "super" models within the same linguistic level as user-defined model information (i.e. the L_1 level). The top linguistic level is therefore no longer a language simply for defining standard metamodels (with no built-in logical levels) but needs to be a fully fledged multi-level modeling language with full support for defining model elements across multiple ontological levels. For this reason we refer to it as the Pan-Level Model (PLM).

3 Implementing an OCA

The OCA has a number of key advantages, including greater overall simplicity [4] and adherence to strict modeling principles in each typing dimension. In effect, all user model information, at all ontological levels, is "instance" data from the point of view of a tool written in terms of the types defined in L_2, the Pan Level Model (PLM). This includes instances of user defined classes, which now occupy a natural place in the hierarchy and are treated no differently to any other user-defined model elements. However, there are also some fundamental problems which need to be overcome to create a practical OCA realization.

3.1 Clabjects and Fields

As can be seen in Figure 1, one of the immediate consequences of a model infrastructure with multiple ontological type levels is that the elements in the central levels are both classes and objects at the same time. Atkinson and Kühne proposed to address this problem by essentially representing all model information in terms of two basic kinds of model elements, clabjects and fields [1]. Fields are the basic value-carrying objects akin to attributes and slots in regular UML. However, in a multi-level modeling environment there is no dichotomy between attributes (value types) and slots (value instances), but rather the unified concept of fields. Clabjects are the more complex entities akin to classes and/or objects in regular UML modeling. However, in a multi-level modeling environment there is also no dichotomy between classes and objects, but the single unified concept of a clabject ("class and object").

Figure 2 shows an example of a simple multi-level model composed of clabjects (e.g. Class, Profession and JobRelation) and fields (Salary). This is nearly the same basic architecture as in Figure 1 but conceptually rotated through 90° in order to correspond to the normal way of representing type levels in UML. At the top level, O_0, we have the abstract conceptual types used in the model together with some domain specific metatypes (e.g. Profession). In the middle level, O_1, we have the domain classes of the kind that are drawn by users in typical UML model scenarios. And at the bottom level, O_2, we have the instances of the user classes.

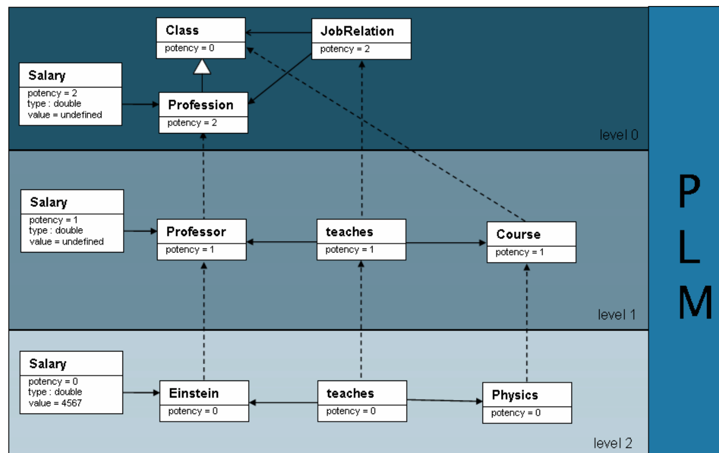

Figure 2 Example Multi-Level Model

3.2 Level and Potency Values

Because there is no longer any built-in distinction between classes and objects or attributes and slots, some other mechanism is needed to indicate whether a model element is intended to represent a type that can be instantiated or a pure instance that is not intended to be instantiated. Atkinson and Kühne proposed the notion of potency to do this [4]. Potency is a single, non-negative Integer value associated with every

model element that indicates how often it can be instantiated. A potency of zero represents a pure instance that cannot be further instantiated while a potency of one represents a normal type that can be instantiated just once. Naturally, the instantiation mechanism requires that an instance have a potency which is one lower than its type. It turns out that in many domains there are situations where types are intended to have an influence beyond their immediate instances (a situation known as deep classification [2]). This situation is naturally handled by allowing clabjects and fields to have a potency greater than 1.

3.3 Uni-Graph Representation of Model Data

The notions of clabjects and fields, augmented with potency and level values, provide the basic tools for representing individual model elements uniformly across multiple levels. However, they do not by themselves ensure that the graphical representation of a model, when interpreted as instances, matches the graphical representation of the model when interpreted as types. In the UML, for example, the graphical representation of a given real world scenario (in terms of nodes and edges) is often different depending on whether the scenario is interpreted as instances of the metamodel (i.e. objects) or as types (i.e. classes)..

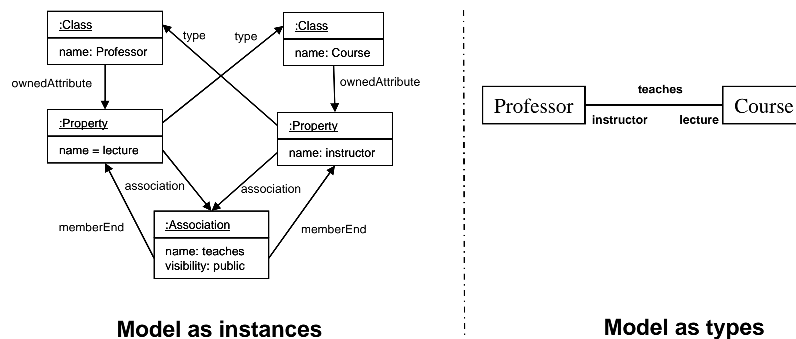

Figure 3 Instance and Type Views of Associated Classes

This is illustrated clearly in Figure 3 which shows how a single scenario (a pair of associated classes) can have two different graphical interpretations in the UML. The left hand side shows an association represented in terms of instances of the UML metaclasses Association and Property etc., and the right hand side shows it represented as types. Both representations are equally valid and use the "official" UML notation for objects and classes, but they have completely different graphical structures. This difference (which cannot be solved by the clabject and field concepts) becomes a problem when a given connector needs to be treated as both a type and an instance in a multi-level modeling infrastructure such as the OCA. In the current UML infrastructure it is not possible to show the type and object views of an association at the same time, because of their different graphical structures

In our realization of the OCA this problem is overcome by separating clabjects into two fundamentally different kinds, *nodes* and *connectors*, and by ensuring that all

logical connections have the same graphical structure whether viewed as types or as instances. This is achieved by representing all associations and links as individual clabjects with a single edge to their source and to their target. This can be seen in Figure 2 where the nodes are Class, Profession, Professor, Course, Einstein and Physics, and the connectors are JobRelation and its instance, teaches. By ensuring that all connections (whether associations or links) are modeled by exactly one clabject, intermediate level connections can be interpreted as both links and associations since they have the same structure.

To retain the UML tradition that associations are represented as individual lines we require connectors to have two renderings – an *exploded* rendering where the clabject is represented using the usual rectangle notation, and an *imploded* rendering where the clabject is represented as a dot (see Figure 4). The motivation for rendering connectors as dots is to retain the visual message that they are nodes (with two edges emanating from them) whilst at the same time giving the traditional visual impression of a single contiguous line from the source class to the target class. This example shows binary connections, but the approach scales up easily to higher order connectors as well.

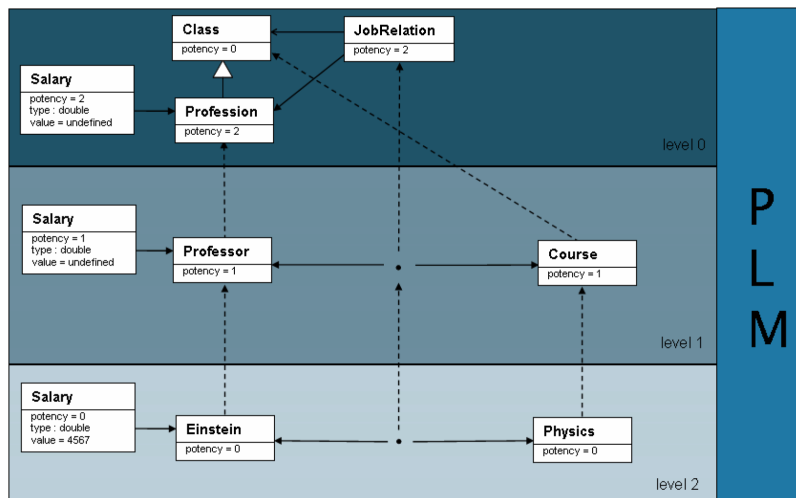

Figure 4 Imploded rendering of Connectors

3 Conclusion

In this position paper we have outlined the key ideas that we believe are needed to realize the OCA and provide flexible access to model information, at all ontological levels, at both development time and run-time. An OCA framework of the kind outlined in this paper can be used to mirror the state of the system in order to check its consistency or as the platform for a model-based run-time system which can support the “execution” of a model. The need to map models to programs or database schemas

will therefore become a thing of the past. Moreover, the coherent and uniform representation of instance, type and metatype information makes it is possible to support intelligent run-time queries and knowledge generation operations of the kind usually associated with ontologies (e.g. subsumption). We therefore believe that the OCA is the basis for a "deep" integration of ontology and modeling technologies. We are currently in the process of creating a prototype implementation of an OCA based on these ideas which includes a run-time repository for storing multi-level model information and a graphical editor for creating and accessing this information.

References

1. Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation, IEEE Software, September/October, pp. 36-41 (2003)
2. Kühne, T. and Steimann, F.: Tiefe Charakterisierung, in B. Rumpe, W. Hesse (editors) Modellierung 2004, LNI 45 (GI, 2004), pp. 121–133, Marburg, March 2004
3. Atkinson, C., Gutheil, M., Kiko, M.: On the Relationship of Ontologies and Models, Second Workshop on Meta-Modelling (WoMM 2006), Karlsruhe, Germany (2006)
4. Atkinson, C., Kühne, T.: Reducing Accidental Complexity in Domain Models, Journal on Software & System Modeling (SoSYM), to appear (2007)
5. OMG: Unified Modeling Language, v1.1. OMG document ad/97-08-04, (1997)
6. OMG: Unified Modeling Language: Infrastructure, v2.1.1. OMG document ad/07-02-06, (2007)