# Definition and Application of Metaclasses[*]

Mohamed Dahchour Alain Pirotte Esteban Zimányi [†]

**Abstract**

Metaclasses are classes whose instances are themselves classes. Metaclasses are generally used to define and query information relevant to the class level. The paper first analyzes the more general term *meta* and gives some examples of its use in various application domains. Then, it focuses on the description of metaclasses. To help better understand metaclasses, the paper suggests a set of criteria accounting for the variety of metaclass definitions existing in the literature. The paper finally presents the usage of metaclasses and discusses some questions raised about them.

## 1 Introduction

Common object models (and languages and database systems based on them) model real-world applications as a collection of objects and classes. Objects model real-world entities while classes represent sets of similar objects. A class describes structural (attributes) and behavioral (methods) properties of their instances. The attribute values represent the object's status. This status is accessed or modified by sending messages to the objects to invoke the corresponding methods. In such models, there are only two abstraction levels: *class level* composed of classes that may be organized into hierarchies along inheritance (i.e., isA) mechanism, and *instance level* composed of individual objects that are instances of the classes in the class level.

[*]In H.C. Mayr, J. Lazansky, G. Quirchmayr, and P. Vogel, editors, Proc. of the 12th Int. Conf. on Database and Expert Systems Applications, DEXA'01, LNCS 2113, pages 32-41, Munich, Germany, 2001. Springer-Verlag

[†]Mohamed Dahchour and Alain Pirotte are with the Université catholique de Louvain, IAG School of Management, Information Systems Unit, 1 Place des Doyens, B-1348 Louvain-la-Neuve, Belgium, e-mail: dahchour@isys.ucl.ac.be, pirotte@info.ucl.ac.be. Esteban Zimányi is with the Université Libre de Bruxelles, Faculty of Engineering, CP 165/15, 50 Av. F. Roosevelt, B-1050 Brussels, Belgium, e-mail: ezimanyi@ulb.ac.be

However, beyond the need for manipulating individual objects, there is also the need to deal with classes themselves regardless of their instances. For example, it should be possible to query a class about its name, list of its attributes and methods, list of its ancestors and descendents, etc. To be able to do this, some object models (e.g., Smalltalk [11], ConceptBase [16], CLOS [18]) allow to treat classes themselves as objects that are instances of the so-called *metaclasses*. With metaclasses, the user is able to express the structure and behavior of classes, in such a way that messages can be sent to classes in the same way that messages are sent to individual objects in usual object models. Systems supporting metaclasses allow to organize data into an architecture of several abstraction levels. Each level describes and controls the lower one.

Existing work (e.g., [11, 16, 18, 19, 26, 10, 21]) only deal with particular definitions of metaclasses related to specific systems. This work deals with metaclasses in general. More precisely, the objectives of the paper are:

- clarify the concept of metaclasses often confused with ordinary classes;
- define a set of criteria characterizing a large variety of metaclass definitions;
- present some uses of metaclasses;
- discuss some problems about metaclasses raised in the literature.

The rest of the paper is organized as follows. Section 2 analyzes the more general term *meta* and gives some examples of its use beyond the object orientation. Section 3 defines the concept of metaclasses. Section 4 presents a set of criteria accounting for the variety of metaclass definitions found in the literature. Section 5 describes the mechanism of method invocation related to metaclasses. Section 6 presents the usage of metaclasses and Section 7 analyzes some of their drawbacks. Section 8 summarizes and concludes the paper.

## 2 Meta Concepts

The word *meta* comes from Greek. According to [29], meta means "occurring later than or in succession to; situated behind or beyond; more highly organized; change and transformation; more comprehensive". Meta is usually used as a prefix of another word. In the scientific vocabulary, meta expresses the idea of change (e.g., metamorphosis, metabolism) while in the philosophical vocabulary, meta expresses an idea of a higher level of generality and abstractness (e.g., metaphysics, metalanguage). In the computing field *meta* has the latter sense and it is explicitly defined as being a "prefix meaning one level of description higher. If X is some concept then meta-X is data about, or processes operating on, X" [15]. Here are some examples of use of meta in computing:

- *Metaheuristic*. It is an heuristic about heuristics. In game theory and expert systems, metaheuristics are used to give advice about when, how, and why to combine or favor one heuristic over another.

- *Metarule*. It is a rule that describes how ordinary rules should be used or modified. More generally, it is a rule about rules. The following is an example of metarule:

  > "If the rule base contains two rules $R_1$ and $R_2$ such that:
  > $R_1 \equiv A \wedge B \Rightarrow C$
  > $R_2 \equiv A \wedge \text{not } B \Rightarrow C$
  > then the expression B is not necessary in the two rules; we can replace the two rules by $R_3$ such that $R_3 \equiv A \Rightarrow C$"

  Metarules can be used during problem solving to select an appropriate rule when conflicts occur within a set of applicable rules.

  Meta-heuristics and meta-rules are known in knowledge-based systems under a more generic term, *metaknowledge*.

- *Metaknowledge*. It is the knowledge that a system has about how it reasons, operates, or uses domain knowledge. An example of metaknowledge is shown below.

  > "If more than one rule applies to the situation at hand, then use rules supplied by experts before rules supplied by novices"

- *Metalanguage*. It is a language which describes syntax and semantics of a given language. For instance, in a metalanguage for C++, the (meta)instruction ⟨variable⟩"="⟨expression⟩";" describes assignment statement in C++, of which "x=3;" is an instance.

- *Metadata*. In databases, metadata means data about data and refer to things such as a data dictionary, a repository, or other descriptions of the contents and structure of a data source [22].

- *Metamodel*. It is a model representing a model. Metamodels aim at clarifying the semantics of the modeling constructs used in a modeling language. For instance a metamodel for OML relationships is proposed in [14]. Figure 1 shows a metamodel for the well-known ER model.

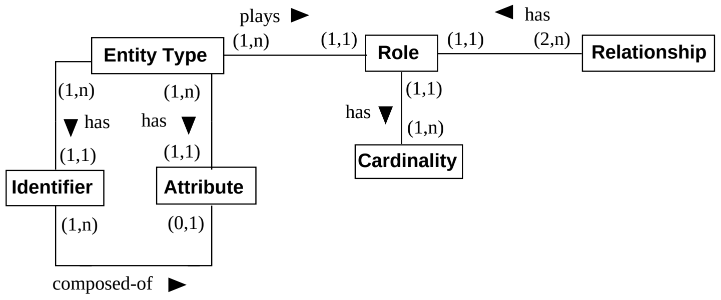

Figure 1: Metamodel of the ER model.

The basic concepts of the ER model are the following: *entity types*, *relationships* associating entity types, *roles* played by participating entity types, *attributes* characterizing entity types or relationships themselves, *identification* structures identifying in a unique manner the entity types, and *cardinalities* related to the roles. Each of these concept appears as a *metatype* in the metamodel shown in Figure 1.

## 3 The Metaclass Concept

In a system with metaclasses, a class can also be seen as an object. *Two-faceted constructs* make that double role explicit. Each two-faceted construct is a composite structure comprising an object, called the *object facet*, and an associated class, called the *class facet*. To underline their double role, we draw a two-faceted construct as an object box adjacent to a class box. Like classes, class facets are drawn as rectangular boxes while objects (and object facets) appear as rectangular boxes with rounded corners as in Figure 2.

MC is a metaclass with attribute A and method M1(..). Object LMC is an instance of MC, with a0 as value for attribute A. LMC is the object facet of a two-faceted construct with C as class facet. A is an *instance attribute* of MC (i.e., it receives a value for each instance of MC) and a *class attribute* of C (i.e., its value is the same for all instances of C). For instances I1_C and I2_C of C, attribute A is either inapplicable (e.g., an aggregate value on all instances) or constant, i.e., an instance attribute with the same value for all instances. In addition to the class attribute A, C defines attribute B and method M2(..). The figure shows that methods like M1(..) can be invoked on instances of MC (e.g., LMC), while methods like M2(..) can be invoked on instances of C (e.g., I1_C and I2_C).

Note that the two-faceted construct above is useful only to illustrate the double facet of a class that is also an object of a metaclass. Otherwise, in practice, both

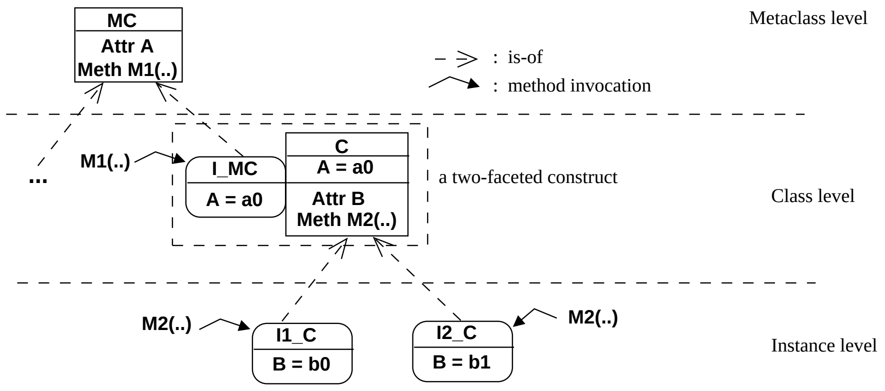

Figure 2: Class/metaclass correspondence.

the object facet I_MC and its associated class facet C (see Figure 2) are the same thing, say, I_MC_C defined as shown in Figure 3.

```
Metaclass MC                  Class I_MC_C instanceOf MC
   Attributes                    Values
        A:AType                       A=a0

   Methods                       Attributes
        M1(..)                        B:BType
End
                                 Methods
                                      M2(..)
                              End
```

Figure 3: Definition of class I_MC_C as an instance of metaclassMC.

Systems with metaclasses comprise at least three levels: token (uninstantiable object), class, and metaclass, as shown in Figure 4. Additional levels, like Metaclass in Figure 4, can be provided as root for the common structure and behavior of all metaclasses. The number of levels of such hierarchies varies from one system to another.

## 4 Various Metaclass Definitions

Substantial differences appear in the literature about the concept of metaclass. We suggest the following criteria to account for the variety of definitions.

- **Explicitness:** the ability for programmers to explicitly declare a metaclass like they do for ordinary classes. Explicit metaclasses are supported by several *semantic models* (e.g., TAXIS [23], SHM [2]), *object models and sys-*

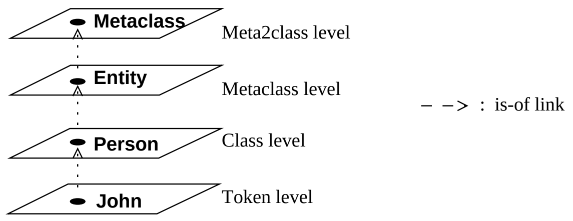

Figure 4: Levels of systems with a metaclass concept.

*tems* (e.g., VODAK [19], ADAM [26], OSCAR [10], ConceptBase [16]), *knowledge representation languages* (e.g., LOOPS [1], KEE [9], PROTEUS [28], SHOOD [25], Telos [24]), and *programming languages* (e.g., CLASSTALK [21], CLOS [18]). On the contrary, Smalltalk [11] and Gemstone [3], for example, only support implicit system-managed metaclasses. Of course, explicit metaclasses are more flexible [21]. They can, for example, be specialized into other metaclasses in the same way that ordinary classes can.

- **Uniformity:** the ability to treat an instance of a metaclass like an instance of an application class. More generally, for a system supporting instantiation trees of arbitrary depth, uniformity means that an object at level *i* ($i \geq 2$), instance of a (meta)class at level *i*+1, can be viewed and treated like an object at level *i*-1, instance of a (meta)class at level *i*. Thus, for example, in Figure 4, to create the Entity metaclass, message new is sent to Metaclass; to create the Person class, the message new is sent to the Entity metaclass; and, again, to create the terminal object John, message new is sent to the Person class. While most metaclass systems support uniformity, Smalltalk-80 and Loops, for example, do not.

- **Depth** of instantiation: the number of levels for the hierarchy of classes and metaclasses. While, for example, Smalltalk has a limited depth in its hierarchy of metaclasses, VODAK and CLOS allow for an arbitrary depth.

- **Circularity:** the ability to use metaclasses in a system for a uniform description of the system itself. To ensure finiteness of the depth of instantiation tree, some metaclass concepts have to be instances of themselves. CLOS and ConceptBase, for example, offer that ability. Smalltalk does not.

- **Shareability:** the ability for more than one class to share the same user-defined metaclass. Most systems supporting explicit metaclasses provide shareability.

- **Applicability:** whether metaclasses can describe classes only (the general case) or other concepts also. For example, TAXIS extends the use of metaclasses to procedures and exceptions, while ConceptBase uses attribute metaclasses to represent the common properties of a collection of attributes.

- **Expressiveness**: the expressive power made available by metaclasses. In most systems, metaclasses represent the structure and behavior of their instances only as shown in Figure 2. In some systems like VODAK [19], metaclasses are able to describe both their direct instances (that are classes) and instances of those classes. The metaformulas of Telos and ConceptBase can also specify the behavior of the instances of a metaclass and of the instances of its instances.

- **Multiple classification**: the ability for an object (resp., class) to be an instance of several classes (resp., metaclasses) not related, directly or indirectly, by the generalization link. At our knowledge, only Telos and ConceptBase support this facility.

Note that this list of characteristics has been identified by carefully analyzing a large set of systems supporting metaclasses. We cannot, however, claim their exhaustiveness. The list remains open to other characteristics that could be identified by exploring other systems. Note also that these criteria are very useful in that they much help designers to select the more suitable system (with metaclasses) to define their specific needs.

## 5 Method Invocation

In systems with metaclasses, messages can be sent to classes in the same way that messages are sent to individual objects in usual object models. To avoid ambiguity, we show below how messages are invoked at each level of abstraction and how objects are created. Henceforth, the term object will denote tokens, classes, or metaclasses. Two rules specify the method-invocation mechanism[1].

*Rule 1.* When message Msg is sent to object o, method Meth which responds to Msg must be available (directly or indirectly by inheritance) in the class of o.

*Rule 2.* An object o is created by sending a message, say new(), to the class of o. Consequently, according to *Rule 1*, new() must be available in the class of o's class.

[1]These rules assume that the target object system represents object behavior with *methods*. Systems like ConceptBase that represent object behavior using constraints and deductive rules are not concerned with message-passing rules.

The following messages illustrate the two rules above. They manipulate objects of Figure 4.

- John$\rightarrow$increaseSalary($1000). In this message, increaseSalary is sent to object John to increase the value of salary by $1000. Method increaseSalary is assumed to be available in the class of John, i.e., Person.
- John := Person$\rightarrow$new(). In this message, new is sent to object Person in order to create object John as an instance of Person. According to *Rule 1*, method new must be available in the class of Person, i.e., Entity.

Most object systems provide for built-in primitives and appropriate syntax to define classes (e.g., Person), their attributes (e.g., salary), and methods (e.g., increaseSalary). However, to illustrate how metaclasses affect classes, just as classes affect tokens, we show in the following how messages can be sent to the Entity metaclass to build classes and their features.

- Person := Entity$\rightarrow$new(). In this message, Person is created as an instance of Entity. Once again, this assumes that method new is available in Entity's class, i.e., Metaclass.
- Entity$\rightarrow$addAttributes(Person, $\{$ [attrName:name, attrDomain: String]; [attrName:salary, attrDomain: Real]$\}$). This message adds attributes name and salary to the newly created object Person. Similarly, a message can be sent to object Entity to add a new method to object Person.

## 6 Usage of Metaclasses

Various reasons warrant a metaclass mechanism in a model or a system. Typically, metaclasses extend the system kernel, blurring the boundary between users and implementors. Explicit metaclasses can specify knowledge to:

- Represent group information, that concerns a set of objects as a whole. For example, the average age of employees is naturally attached to an EmployeeClass metalevel.
- Represent class properties unrelated to the semantics of instances, like the fact that a class is concrete or abstract[2], has a single or multiple instances, has a single superclass or multiple superclasses.

[2]Here, an abstract class, in the usual sense of object models, is an incompletely defined class without direct instances, whose complete definition is deferred to subclasses.

- Customize the creation and the initialization of new instances of a class. The message new which is sent to a class to create new instances can incorporate additional arguments to initialize the instance variables of the newly created instance. Furthermore, each class can have its own overloaded new method for creating and initializing instances.

- Enhance the extensibility and the flexibility of models, and thus allow easy customization. For example, the semantics of generic relationships can be defined once and for all in a structure of metaclasses that provides for defining and querying the relationships at the class level, creating and deleting instances of participating classes, and so on (see e.g., [13, 19, 5, 7, 20, 6]).

- Extend the basic object model to support new categories of objects (e.g., remote objects or persistent objects) and new needs such as the authorization mechanism. This kind of extension requires the ability to modify some basic behavioral aspects of the system (object creation, message passing), and has often been faced by allowing these aspects to be manipulated in a metaclass level.

- Define an existing formalism or a development method within a system supporting metaclasses. This definition roughly consists in representing the modeling constructs involved in that formalism or method (i.e., its ontology) with a set of metaclasses of the target system. For example, Fusion [4], an object development method, was partially integrated in ConceptBase [12] using metaclasses.

- Integrate heterogeneous modeling languages within the same sound formalism. For example, a framework combining several formalisms for the requirement engineering of discrete manufacturing systems was defined along the lines of ConceptBase in [27]. The combined formalisms are: CIMOSA (for the purpose of eliciting requirements), i* (for the purpose of enterprise modeling), and the Albert II language (for the purpose of modeling system requirements).

## 7 Problems with Metaclasses

Some authors (e.g., [17]) have pointed out some problems with metaclasses. These problems have been analyzed in part in [8]. We summarize the main issues.

- *Metaclasses make the system more difficult to understand.* We agree with [8] that, once programmers are familiar with metaclasses, having a single mechanism for both data and metadata helps them progress from object design to object *system* design.

- *By themselves, metaclasses do not provide mechanisms to handle all the run-time consequences of extending the data model.* This is true for most systems. However, some systems like ADAM [8] and ConceptBase introduce the notion of active rules to enforce some constraints in order to keep the database in a consistent state.

- *Metaclasses do not facilitate low-level extensions.* For most systems this is true since metaclasses describe the model or class level, above the structures that specify storage management, concurrency, and access control. Thus, in such systems, metaclasses do not let applications define policies at all levels. However, this is not a general rule. In fact, systems such as ConceptBase and VODAK provide for a mechanism of metaclass that allows to describe both the class and instance level in a coordinated manner.

- *With metaclasses, programmers must cope with three levels of objects: instances, classes, and metaclasses.* We agree that it can be difficult at the beginning to play with the three levels.

After presenting these problems, the authors conclude that the metaclass approach is not satisfactory. We agree with [8] that this conclusion may be be valid when talking about programming languages, but we believe that explicit metaclasses are a powerful mechanism for enhancing database extensibility, uniformity, and accessibility by addressing these issues at the class level (see e.g., [6]).

## 8 Conclusion

Metaclasses define the structure and behavior of class objects, just as classes define the structure and behavior of instance objects. In systems with metaclasses, a class can also be seen as an object. We used the two-faceted constructs to make that double role explicit. Substantial differences appear in the literature about the concept of metaclass. We suggested a set of criteria to account for the variety of definitions, namely, uniformity, depth of instantiation, circularity, shareability, applicability, and expressiveness. We then presented the method-invocation mechanism between objects at various levels of abstraction. We also presented some uses of metaclasses and analyzed some of their drawbacks pointed out in the literature.

## References

[1] D.G. Bobrow and M.J. Stefik. *The LOOPS Manual*. Xerox Corp., 1983.

[2] M.L. Brodie and D. Ridjanovic. On the design and specification of database transactions. In M. L. Brodie, J. Mylopoulos, and J. W. Schmidt, editors, *On Conceptual Modelling*. Springer-Verlag, 1984.

[3] P. Butterworth, A. Ottis, and J. Stein. The Gemstone Database Management System. *Communications of the ACM*, 34(10):64–77, 1991.

[4] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object-Oriented Development: The Fusion Method*. Prentice Hall, 1994.

[5] M. Dahchour. Formalizing materialization using a metaclass approach. In B. Pernici and C. Thanos, editors, *Proc. of the 10th Int. Conf. on Advanced Information Systems Engineering, CAiSE'98*, LNCS 1413, pages 401–421, Pisa, Italy, June 1998. Springer-Verlag.

[6] M. Dahchour. *Integrating Generic Relationships into Object Models Using Metaclasses*. PhD thesis, Département d'ingénierie informatique, Université catholique de Louvain, Belgium, March 2001.

[7] M. Dahchour, A. Pirotte, and E. Zimányi. Materialization and its metaclass implementation. To be published in IEEE Transactions on Knowledge and Data Engineering.

[8] O. Díaz and N.W. Paton. Extending ODBMSs using metaclasses. *IEEE Software*, pages 40–47, May 1994.

[9] R. Fikes and J. Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9), September 1985.

[10] J. Göers and A. Heuer. Definition and application of metaclasses in an object-oriented database model. In *Proc. of the 9th Int. Conf. on Data Engineering, ICDE'93*, pages 373–380, Vienna, Austria, 1993. IEEE Computer Society.

[11] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.

[12] E.V. Hahn. Metamodeling in ConceptBase - demonstrated on FUSION. Master's thesis, Faculty of CS, Section IV, Technical University of München, Germany, October 1996.

[13] M. Halper, J. Geller, and Y. Perl. An OODB part-whole model: Semantics, notation, and implementation. *Data & Knowledge Engineering*, 27(1):59–95, May 1998.

[14] B. Henderson-Sellers, D.G. Firesmith, and I.M. Graham. OML metamodel: Relationships and state modeling. *Journal of Object-Oriented Programming*, 10(1):47–51, March 1997.

[15] D. Howe. *The Free On-line Dictionary of Computing*. 1999.

[16] M. Jarke, R. Gallersdörfer, M.A. Jeusfeld, and M. Staudt. ConceptBase : A deductive object base for meta data management. *Journal of Intelligent Information Systems*, 4(2):167–192, 1995.

[17] S.N. Khoshafian and R. Abnous, editors. *Object Orientation: Concepts, Languages, Databases, User Interfaces*. John Wiley & Sons, New York, 1990.

[18] G. Kiczales, J. des Rivières, and D. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[19] W. Klas and M. Schrefl. *Metaclasses and their application*. LNCS 943. Springer-Verlag, 1995.

[20] M. Kolp. *A Metaobject Protocol for Integrating Full-Fledged Relationships into Reflective Systems*. PhD thesis, INFODOC, Université Libre de Bruxelles, Belgium, October 1999.

[21] T. Ledoux and P. Cointe. Explicit metaclasses as a tool for improving the design of class libraries. In *Proc. of the Int. Symp. on Object Technologies for Advanced Software, ISOTAS'96*, LNCS 1049, pages 38–55, Kanazawa, Japan, 1996. Springer-Verlag.

[22] L. Mark and N. Roussopoulos. Metadata management. *IEEE Computer*, 19(12):26–36, December 1986.

[23] J. Mylopoulos, P. Bernstein, and H. Wong. A language facility for designing interactive, database-intensive applications. *ACM Trans. on Database Systems*, 5(2), 1980.

[24] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about informations systems. *ACM Trans. on Office Information Systems*, 8(4):325–362, 1990.

[25] G.T. Nguyen and D. Rieu. SHOOD: A desing object model. In *Proc. of the 2nd Int. Conf. on Artificial Intelligence in Design*, Pittsburgh, USA, 1992.

[26] N. Paton and O. Diaz. Metaclasses in object oriented databases. In R.A. Meersman, W. Kent, and S. Khosla, editors, *Proc. of the 4th IFIP Conf. on Object-Oriented Databases: Analysis, design and construction, DS-4*, pages 331–347, Windermere, UK, 1991. North-Holland.

[27] M. Petit and E. Dubois. Defining an ontology for the formal requirements engineering of manufacturing systems. In K. Kosanke and J.G. Nell, editors, *Proc. of the Int. Conf. on Enterprise Integration an Modeling Technology, ICEIMT'97*, Torino, Italy, 1997. Springer-Verlag.

[28] D.M. Russinof. Proteus: A frame-based nonmonotonic inference system. In W. Kim and F.H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 127–150. ACM Press, 1989.

[29] M. Webster. *The WWWebster Dictionary*. 2000.