RMondel: A Reflective Object-Oriented Specification Language

(Extended Abstract)

M.Erradi, G.v.Bochmann

Département d'Informatique et de Recherche
Opérationnelle
Université de Montréal
C.P. 6128, Succ. "A", Montréal,
P.Q., Canada, H3C-3J7
E-mail: {erradi bochmann } @iro.umontreal.ca

Abstarct: In this paper we describe our work in the computational reflection area. We first discuss the motivation that leads us to the choice of reflection which we use to enhance our object-oriented language *Mondel*. After a brief introduction to *Mondel*, we discuss *RMondel* (Reflective *Mondel*) which was designed to help developing modifiable specifications.

1. Introduction and motivations

Our goal is to design support for modifiable specifications by enhancing the language *Mondel*, an object-oriented database specification language [Boch 89a]. Looking at the database aspect of this language, modifying a specification means altering the structure of a database schema [Bane 87],[Skar 86],[Rous 85]. One step in this direction, is to look for developing a meta-level structure which consists of modeling the basic structure of the language which is that of a type (called "class" in most object-oriented languages). This modeling process leads us to defining the structure of certain meta-concepts and constraints that must hold to allow a smooth transition between the actual state of a system and its new state after the modification of the specification. The system must transit to a consistent state with respect to the previous one.

As our starting language *Mondel* is object-oriented, our need can be formulated as follows: In order to allow users to make some type structure alterations within a runing system, we have to enhance the object model to provide a uniform treatment within a system by considering that a type is an object of type TYPE. This consideration provides a uniform treatment for object attributes and type parameters for generic types. This is useful in the context of dynamically changing type definitions, as considered in the context of systems evolving over long periods of time [Card 86], [Bane 87],[Skar 86],[Rous 85].

To allow users the construction of modifiable specifications, we have developped *RMondel*, an extension of *Mondel*, to provide means for assisting the evolution of specifications. One step in this direction is to consider that a type is an object of type *TYPE* [Erra 90], and to provide a uniform treatment for types and other objects in the language. This leads us to adapt the reflection technique [Maes 87] in a maner similar to ObjVLisp [Coin 87] and ObjVProlog [Male 90]. In contrast to the latter approaches, *RMondel* supports object persistence, and considers not only types as objects, but also operations (methods), attributes and the behavior. The fact to consider the behavior, language constructs, as objects is also considered in KSL approach [Ibra 88].

2. Mondel overview

We have developed an object-oriented database specification language, called *Mondel* [Boch 89a], particulary suitable for specifying applications in the network management area. *Mondel* has certain interesting features, such as multiple inheritance, type checking, rendez-vous communication between objects, the possibility of concurrent activities performed by a single object and object persistence. *Mondel* has also a formal semantics, expressed by means of a translation to a state transition system, which is the basis for the verification of *Mondel* specifications, and for the construction of an interpreter.

Each *Mondel* object has an identity, some attributes, some operations, and a behavior which provides certain details as constraints on the order of execution of operations by the object. The *Mondel* object model is appropriate for defining small objects such as integers, characters and booleans as well as large objects such as a whole system (i.e. a compiler). Each Mondel object is of a given type. A type definition specifies the

properties of an object type and each created object of that type has all properties defined for the type.

Moreover, types can be related to each other by means of inheritance. It is important to note that for many authors the concept of inheritance is only concerned with the names and parameter types of the operations offered by the specified object type [Blac 87], [Card 88], [Meye 88]. However, there are other important aspects to inheritance which considers comparing the dynamic behavior of objects [Boch 89b], including constraints on the results of operations, the ordering of operation execution, and the possibilities of blocking.

Mondel is a strongly typed language. It uses a type checking mechanism based on type compatibility. Type checking is important for the construction of correct specification, and it allows the detection of a large fraction of specification errors during the compilation phase. Mondel imposes type checking for effective parameters for called operations, attribute assignments during object creation and similar situations.

3. *RMondel* : Reflective Mondel

The principle of considering types as objects within a system, leads us to look at reflection and reflective architectures as a promising choice. Reflective facilities has shown a powerful expressiveness, while they encourage modular descriptions of computation, for several object-oriented programming languages [Maes 87],[Ibra 88],[Coin 87],[Yone 89],[Male 90]. The reflection principle was defined by Pattie Maes in [Maes 87] as the behavior of a reflective system, where a reflective system is a computational system able to act upon itself.

In conventional computational systems, computation is performed on data that represent entities which are external to the system. In contrast, a reflective computational system must contain some data which represent the structural and computational aspects of itself, and such data must be accessible and modifiable within the system itself, and hopely, the changes made to such data must be causally reflected in the actual computations being performed. The great interest of computational reflection is to allow a dynamic modification of the internal organization of the system.

We have developped *RMondel*, an extension of *Mondel*, to adapt the reflection technique [Maes 87] in a maner similar to ObjVLisp [Coin 87] and ObjVProlog [Male 90]. In contrast to the latter approaches, *RMondel* supports object persistence, and considers not only types as objects, but also operations (methods), attributes and the behavior.

The architecture of *RMondel* is supported by two graphs: The instanciation graph and the inheritance graph. The instanciation graph represents the "instance of" relationship, and the inheritance graph represents the "subtype of" (inheritance) relationship. *TYPE* and *OBJECT* are the respective roots of these two directed graphs shown in Figure 1.

The general structure of *RMondel* is obtained by the connection of these two graphs. The initial state of the system contains the objects: *TYPE, OBJECT,Attribute, Operation* and others. Some elements of the *RMondel* model, such as attributes, operations, inheritance, behavior and procedures, are considered as objects and instances of specific types, for example the attributes of a type, or those of an instance of such a type, are instances of the type *Attribute*. Since types are objects, user defined types and the above mentioned system types are instances of the type *TYPE* which is also an instance of itself.

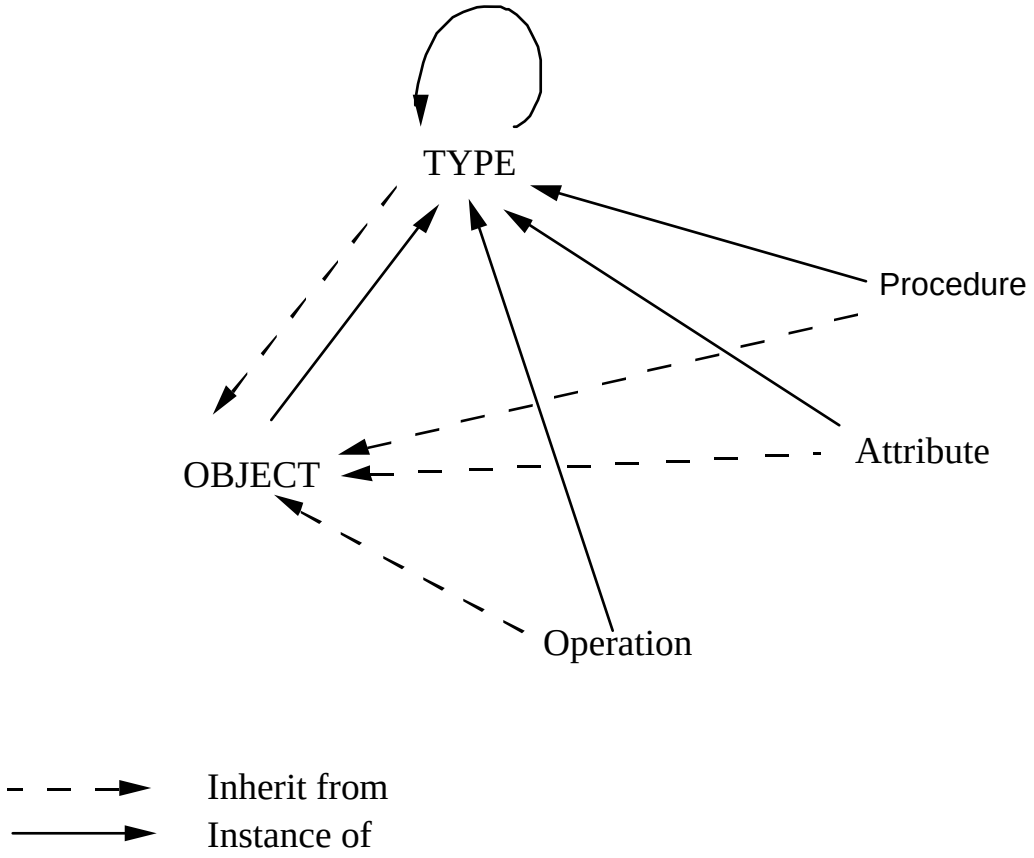

Figure 1. A model of RMondel (Simplified)

Figure 1 shows the instanciation graph superposed with the inheritance graph. *TYPE* is the type which holds the default behavior for types as objects. It defines the ***New*** operation which creates a new object. We have also, a meta operation called ***NewAttr*** to create an attribute for an object (instance as well as type definition). *OBJECT* is the type which defines the default behavior for all object instances. *Attribute* is the type which defines the default behavior of attributes as objects. *Operation, Behavior* and *Procedure* are the types which defines the default behavior of operations, behavior statements and procedures as objects respectively.

4. Conclusion

We have presented an overview of a reflective object-oriented specification language where we consider not only types as objects, but also operations (methods), attributes and

the behavior. Moreover, many aspects of the application of such a model are the subject of on going work. A prototype of *RMondel* is currently under development.

References

[Bane 87] J. Banerjee, W. Kim, H. J. Kim and H. F. Korth, *Semantics and implementation of schema evolution in object oriented databases*, in Proceedings, ACM SIGMOD Int. Conf. On Management of Data, San Fransisco, CA, May 1987, pp. 311-322.

[Blac 87] A. Black, N. Hutchinson, E. Jul, H. Levey and L. Carter, *Distribution and abstract types in Emerald*, IEEE Trans. on Soft. Eng., Vol SE-13, no.1,1987, pp.65-76.

[Boch 89a] G. v. Bochmann, M. Barbeau, A. Bean, M. Erradi and L. Lecomte, *Object-oriented databases: Modelling and specification of applications in the field of network management*, Report for research contract CRIM/BNR, April 1989.

[Boch 89b] G. v. Bochmann, *Inheritance for objects with concurrency*, Publication departementale # 687, Departement IRO, Université de Montréal, Septembre 89.

[Card 86] L. Cardelli, *A polymorphic lambda-calculus with Type:Type*, Technical Report No. 10, Digital System Research Center, 130 Lyton avenue, Palo Alto, CA 94301, May 1986.

[Card 88] L. Cardelli, *A semantics of multiple inheritance*, Information and Computation 76 (1988), pp. 138-164.

[Coin 87] P. Cointe, *Metaclasses are first class: The ObjVLisp Model*, OOPSLA'87, ACM Sigplan Notices 22, 12, pp.156-167.

[Erra 90] M. Erradi and G. v. Bochmann, *Definition de RMondel*, rapport technique, Département IRO, Université de Montréal.

[Ibra 88] M. H. Ibrahim and F. A. Cummins, *KSL: A Reflective Object-Oriented Programming Language*, IEEE, Proceedings of the Int. Conf on Computer Languages. 1988, pp.186-193.

[Maes 87] P. Maes, *Concepts and Experiments in computational reflection*, OOPSLA'87, ACM Sigplan Notices 22, 12, pp.147-155.

[Male 90] J. Malenfant, *Conception et implantation d'un langage de programmation logique, par objets et repartie*, Dept. IRO. Université de Montréal, Janvier 1990.

[Meye 88] B. Meyer, *Object Oriented Software Construction*, C.A.R. Hoare Series Editor, Prentice Hall, 1988.

[Rous 85] N. Roussopoulos and L. Mark, *Schema Manipulation in Self-Describing and Self-Documenting Data Models*, Int. Journal of Computer and Information Sciences, vol. 14, no. 1, 1985.

[Skar 86] A. H. Skarra and S. B. Zdonik, *The Management of Changing Types in an Object-Oriented Database*, OOPSLA'86 Proceedings, september 1986.

[Yone 89] A. Yonezawa and T. Watanabe, *An Introduction to object-based reflective conccurent computation*, ACM Sigplan Notices, Vol.24, No.4, 1989, pp.50-54.