Packaging Knowledge into Metaobjects

David Edmond, Mike Papazoglou, Nick Russell and Zahir Tari
School of Information Systems, Queensland University of Technology
GPO Box 2434 Brisbane Queensland 4001 Australia
email: {davee,mikep,nickr,zahirt}@icis.qut.edu.au

Abstract

The use of reflection [Mae87, HY88, Pae90] is particularly applicable to multi-database systems and to cooperating systems in general. We view such systems as (1) being distributed over a common communication network, and (2) working towards some common goal. Cooperation is achieved by coordinating and exchanging information and expertise. Conventional database systems are *not* cooperative: the knowledge they contain is inaccessibly buried within application code.

In [EPT95], we discuss the R-OK Model and suggest that this problem may be overcome by surrounding each local database system with a layer of special reflective metaobjects. The term *meta* object is used only to indicate the relation of such an object to the object it describes. A metaobject is just another object, with structure and behaviour. These objects are used to capture domain and operational knowledge, and to describe, at least in part, remote systems and to monitor task-oriented activities. In this way, we can turn interconnected conventional database systems into a set of cooperating knowledge-based systems. In the R-OK model, every object has access to four metaobjects:

1. A `state` metaobject knows the structure of any associated object, naming each attribute and specifying its type. For example, if the application domain was a simple savings bank, then a savings object might be described by a `state` metaobject as having an account Id, a balance and a minimum-balance-this-month attributes. By its nature, such a metaobject provides only a static picture of an object.

2. A `can` metaobject knows about the behaviour of any associated object – it knows what an object *can* do. This object may also be associated with a number of domain objects, all of which share the same (outward) behaviour. In this metaobject, activities are described in terms of pre- and post-conditions. In the bank example, the post-condition of the *Withdraw* methods might require that if the new balance is less than the previous minimum, then the minimum is reset. Such a metaobject allows a system to consider possible behaviour and its consequences to the object(s) concerned. It also allows a system to investigate alternative ways of achieving some goal. Should it be necessary, for example, to increase the balance of an account, it may be that there are two ways of accomplishing this – either through a conventional deposit *or* by applying interest to the account.

3. A `loc` metaobject knows how to *locate* attributes and *execute* the methods of an object. This metaobject contains:

- A *Lookup* table which indicates how each attribute of the associated domain object is materialised. This reification is accomplished by *surrogate* objects. These metalevel objects have specific knowledge of the location of data.
- A *Do* table which contains, for each method, procedural descriptions of how that method is effected. Should an interpreter be used to execute such code, it will use the *Lookup* table to resolve symbols that it does not recognise.

4. An `act` metaobject knows about the *activity* in which some group of objects is involved. It is a task-oriented object that monitors the activities of the collection of objects that constitute *its* domain.

Reflection, by means of these four metaobjects, not only allows descriptions of the capabilities of existing information systems and their inter-relationships but also facilitates the specification and implementation of a new system by means of composition, that is, by drawing upon the functionality of existing systems.

Because a metaobject is just another object, with structure and behaviour, we may ask whether it too has access to descriptions of itself. In [EPT94], we use these constructs to penetrate aspects of information systems that are usually closed to us; on particular, we look at two examples of how knowledge of behind-the-scene actions may be used to enable cooperation.

In this presentation, we will discuss how the model may be used to provide translations from an object-oriented model into a relational database.

References

[EPT94] Edmond D., Papzoglou M. and Tari Z. (1994) "Using Reflection as a Means of Achieving Cooperation", Procs of FGCS'94 Workshop on Heterogeneous Cooperative Knowledge-Bases, Tokyo.

[EPT95] Edmond D., Papzoglou M. and Tari Z. (1995) "R-OK: A Reflective Model for

Distributed Object Management", Procs of RIDE'95 (Research Issues in Data Engineering), Taiwan.

[HY88] Y. Honda and A. Yonezawa, "Debugging Concurrent Systems Based on Object Groups", Procs of ECOOP'88: European Conference on Object-oriented Programming, Oslo, Norway.

[Mae87] Maes P. (1987). "Concepts and Experiments in Computational Reflection", OOPSLA'87.

[Pae90] Paepcke A. (1990). "PCLOS: Stress Testing CLOS", OOPSLA'90.