Mapping Object-Oriented Concepts into Relational Concepts by Meta-Compilation in a Logic Programming Environment

Johann Christoph Freytag, Rainer Manthey, Mark Wallace
European Computer-Industry Research Centre
Arabellastr. 17
D-8000 München 81, West Germany

Abstract
In this paper we discuss implementation aspects of how to map object-oriented concepts into relational concepts to achieve both, flexibility and efficiency, at the same time. Our past experience shows that such a mapping should not be fixed, but should be flexible for different reasons [Wa186], [B*86b]. On the other hand to ensure an efficient evaluation of requests on the object level, the mapping onto the relational level should not involve any major overhead. We argue that methods of meta-compilation provide an adequate basis for achieving both goals simultaneously [SS86]. In particular, we argue that these techniques are easily implementable in a logic programming environment.

1 Introduction

Over the last years efforts to provide concepts and to implement systems that go beyond relational technology, have focused on two main areas: the extension of relational technology by object-oriented concepts and the combination of logic programming and database technology.

The extension of database technology by object-oriented concepts has been motivated by the need for more expressive data models. Object-oriented data models have been developed e.g. for PROBE [MD86], IRIS [L*86], [LV87], and other systems. The GemStone database system successfully merges database technology into the Smalltalk language [MS86]. More reports on results of the increased (and diverse) research activities to merge object-oriented concepts with database technology can be found in [DD86].

At the same time there has been an increased interest in merging database technology with logic programming concepts. The research work at MCC and at ECRC in particular has focused
on combining concepts of both areas. This effort has also recognized the need to provide more powerful data modeling capabilities than what is present in today's logic programming environments. Galiere emphasized the need to "extend the logic programming paradigm with the object-oriented one" [Gal87]. MCC's languages LDL [TZ86] and FAD [B*86a] include object-oriented concepts. ECRC's KB2 system provides an object-based user language enriched with inheritance and extensive capabilities to manipulate both, data and metadata [Wal86].

Despite all these efforts, little is known about how to map object-oriented concepts into more traditional ones, especially into relational concepts, in a systematical, flexible, and efficient manner. Our own experience with KB2 has made apparent the need for a flexible mapping that can easily be changed depending on various, often conflicting requirements [Wal86]. Lyngbaek and Vianu report on a formal mapping of the IRIS data model into the relational model that serves as the basis for the IRIS implementation [LV87]. However, their work does not discuss the implementation-related aspects of their mapping. In [DKM85], Farmer, King, and Myers describe basic building blocks that help to implement object-oriented "schema" specifications. Valduriez et al. address the mapping problem for complex objects by proposing the N-ary storage model (NSM) and the decomposition storage model (DSM) [V*86].

2 The Approach

Independent of the details of the object-oriented data model, it seems to be generally agreed to represent object-oriented concepts by relational ones and to map operations on objects into operations on relations, respectively. The major advantage of this choice is that one can store data and evaluate queries using existing, well understood relational technology.

However, with this general approach there does not seem to exist one agreed, unique mapping from the object to the relational level that satisfies all possible requirements. We therefore argue that a clean separation between the mapping specification and the use of the mapping is necessary and desirable to accommodate different mappings for different object/operation definitions, e.g. NSM and DSM as in [V*86]. This separation leads to two distinct "pieces": the mapping specification and the "interpreter" of this mapping (for objects and operations) from the object level to the relational level (see Figure 1).

Based on this mapping model, we view the interpreter as a function $I_{OR}(MS_{OR}, RO) \rightarrow RR$ that produces a request $RR$ on relations given the initial request $RO$ on objects, and the mapping specification $MS_{OR}$. Notice, that the request generated might include operations that are not present in the initial object request. For example, the mapping might generate operations that query "schema information" or check integrity constraints that need to be satisfied for a correct object representation by relational concepts.

Despite the aim of separating the mapping specification from its execution for flexibility reasons, one would prefer a mapping component $MO_{OR}(RO) \rightarrow RR$ that tightly intertwines the two aspects,