Beyond Data Dictionaries: Towards a Reflective Architecture of Intelligent Database Systems *

Rainer Manthey

Department of Computer Science III
University of Bonn
Römerstraße 164, D-53117 Bonn, Germany
e-mail: manthey@informatik.uni-bonn.de

Abstract. The main conjecture of this contribution is that forthcoming intelligent database systems - in particular future DOOD systems - should be designed in such a way that a major part of the services they provide are implemented using these same services in a bootstrapping-like manner. We call such an approach "reflective", as is often done by researchers in AI and programming languages. Data dictionaries, being part of any reasonable database system today, exhibit the reflective principle in a nutshell, if they are implemented by means of the same data structures that hold application data. However, even for data dictionaries the reflective implementation is often abandoned for performance reasons. Applying reflection for more advanced and ambitious purposes, up to integrity control or query optimization, is viewed even more skeptically by many, despite the conceptual elegance of the approach. On the other hand, there are a few successful approaches around today that can be interpreted as exhibiting a reflective nature. It is the purpose of this paper to identify such examples and to encourage research to invest more in the reflective style and to look for new solutions to the obstacles still ahead.

1 Introduction

The notion of an "intelligent database" is currently gaining increasing popularity within the database research community. There are two new international journals referring to intelligent databases in their name. A new series of international conferences on intelligent and cooperative databases has been launched. Workshops and courses, talks and articles on this topic can be encountered more and more frequently, and - last not least - a call for project proposals on intelligent databases was issued by the Commission of the European Community as part of their 1992 ESPRIT workprogramme. Like many other notions discussed in research these days, the term "intelligent database" is open to a wide range of interpretation. Rather than having been introduced on purpose by an individual person along with a fixed definition (as was the case for the relational or the ER model, e.g.), intelligent databases have been emerging gradually and unsystematically, thus reflecting the joint intuition of a group of researchers that certain aspects of DB technology are approaching a kind of new quality.

* This work was supported in part by the Commission of the European Community under ESPRIT project number 6333 (IDEA); however, this paper does not constitute a statement of the IDEA consortium, but represents the author's opinions only
Numerous proponents of this notion have been interpreting the qualification "intelligent" in terms of the kind of functionality offered at the user interface of a database system. Both, the way how knowledge can be represented and retrieved as well as the means of its manipulation can thus be qualified as (more or less) intelligent. It is not surprising that DOOD functionality - rules, objects, constraints and the like - has been rather frequently proposed as candidate features of such intelligent DB interfaces. However, there is another, complementary way how the term "intelligent database" can be interpreted, namely as a qualification of the internal organization of the respective DBMS. Although intuitively everybody will agree that a high degree of intelligence has to be incorporated into the sophisticated software modules from which a DBMS is composed (such as compilers, optimizers, storage managers etc.), it is by no means clear that the way how this system software itself is organized corresponds to the criteria defining "intelligence" at the interface of the very system.

It is the intention of this contribution to plead for an elegant and conceptually attractive way of designing future database systems in a more intelligent way. We believe that a wide range of knowledge built into DBMS components can be very adequately represented in terms of those features that an intelligent DB interface offers itself: facts, rules (both active and passive), objects, constraints, class hierarchies, attributes and so on. For sure such a representation is a nice device for formal studies and tutorial purposes. However, it is a possible paradigm of implementation as well, and it is this style of system implementation that we would like to suggest as a serious alternative here. We call this principle "architectural reflection": using the functionality realized at the interface of a system for implementing major parts of the system itself.

Of course, such a reflective architecture is feasible only if there remains a core of "irreducible" procedures from which more complex systems can be systematically bootstrapped. These core procedures would be applied to a basic collection of predefined knowledge available in the internal knowledge base of a reflective system. Identifying what exactly constitutes a core procedure, and how to represent the knowledge required for system bootstrapping in the most appropriate form is an exiting open problem, which is far from even a preliminary solution to date.

The idea of a reflective architecture, or reflective software in general, is absolutely not new in computer science. There are reflective paradigms such as compile-time reflection in programming languages, or meta-classes in object-oriented programming. AI systems organized in terms of extensive internal knowledge bases have been investigated since long. However, in the database community a conscious use of a reflective organization is still the exception rather than the rule.

At the very core of every attempt to realize database system software by means of database concepts is the old idea of organizing a database schema as a meta-database, more precisely: as an instance of a meta-schema expressing the concepts of a particular data model. This kind of primitive internal database of a DBMS is nowadays mostly called data dictionary. Already in the earliest specifications of SQL, the intention to manipulate and query schema data in terms of SQL itself are clearly stated, based on an implementation of the data dictionary of a relational system in terms of special system relations. In many systems even this very early step towards a reflective architecture has not been realized, mostly because of performance reasons.

In contrast to this reaction, there is a recent trend in object-oriented database systems to extend the data dictionary into a full-fledged meta-database, treating not only classes