EXTENDING OBJECT-ORIENTED DATABASES WITH RULES

P. Soupos, S. Goutas, D. Christodoulakis

Computer Engineering Dept.
and
Computer Technology Institute
University of Patras, 26500 Patras, Greece

In the recent years we have seen a substantial influx of ideas in the DBMS technology coming from object-oriented programming languages, logic programming and rule based systems. This stimulated a lot of research effort in the area and a number of object-oriented models and systems have been put forward [Kim89]. A proposal that seems to attract attention recently is the integration of object-oriented and deductive databases [Ball88]. In this abstract we investigate the use of rules in an object-oriented DBMS. We focus on issues of structure, integrity and retrieval. Even though our purpose is not to tackle issues of an object-oriented knowledge based management system but rather to deal with the incorporation of rules in an object-oriented DBMS, the proposed system can be used as a knowledge base because it provides objects for describing complex structured data, rules for expressing object-dependent constraints and object associations, and finally, logic for inference and retrieval. Furthermore our approach can be used in connecting knowledge bases by transforming the data models of the knowledge bases to be connected, to the one proposed here. Interconnection is achieved by associating objects belonging to different knowledge bases via rules.

A major issue in the O-O DBMS's nowadays is the degree to which knowledge is explicit or embedded. Knowledge concerning object types is expressed mainly in the form of methods and in that sense it is embedded. When, for instance, attributes and methods are used to express relationships between objects, this representation lacks qualities such as naturalness, simplicity, uniformity, and understandability. Rules however, which are the most popular form of knowledge representation in expert systems, can serve as a vehicle for expressing functionalities of objects declaratively with all the previously mentioned attractive features.

Another major issue that emerges from knowledge representation is that of integrity constraints. Most existing O-O systems [Kim89] typically rely on the inherent constraints of the type system to assure consistency in the sense that a type constraints the permissible methods that can be applied to an object of that type. To overcome the limited constraint specification facility provided by inherent constraints alone O-O systems must be provided with a mechanism for specifying explicit constraints. Such constraints express explicitly value ranges, dependence, referential integrity and any other type of restriction that cannot be expressed by the type system, for any kind of method and not only for update. Here again rules can give an answer since in typical "if X then Y" rules, X must be true for a situation in order to infer action Y.

Yet another well recognized problem is that the role of declaratively specified knowledge in the form of constraints must be reexamined in the sense that in an object-oriented environment where any operation may be user defined, the system cannot automatically determine which operations can see the effects of which others, or what the side-effects of a given operation might be. This comes as a result of the absence of explicit connection between constraints and the pertaining operations. In order to minimize checking as many constraints, related to an operation, as possible should be in the same description as the operation itself. The formalism of rules again provides the solution.

Based on the issues raised in the preceding discussion we have adopted rules as a means of expressing knowledge which is uniform throughout the type lattice but needs to be customized for each object type with integrity constraints. More specifically such knowledge involves associations between instances, between objects and between objects in different databases, grouping of instances, creation and deletion of object instances and any other activity that requires a uniform approach in order to assure consistency, and transparency.

Finally based on the representation of associations between objects, which as already mentioned can be made by rules, we have used a logical query language. Our goal was not a full-fledged object-oriented programming language but rather a logical query language with base predicates associations between objects. A logical query language has the obvious advantage of expressiveness and simplicity, but poses certain difficulties in evaluation. Evaluation is an issue in the case of recursive queries due to time-consuming join operations. We overcome this problem successfully by transforming recursion into iterative navigation through associations between object instances.

Now we will discuss briefly the basic concepts of our data model, more can be found in [Gout89, Chri89]. This model is composed of what we consider essential structural components of representation in application areas such as CAD/CAM, office information, AI etc. We have focused on simplicity and have used as few basic constructs and concepts as possible. After all the abundance of concepts with some of them overlapping is one of the major criticisms against object-oriented systems.

The need for O-O systems to support sets is well recognized [OOPS87]. In O-O database systems based on languages that provide persistence, sets are considered as extensions of types and it is up to the programmer to decide what is to be kept and what is not. In the case of databases though, sets must be managed automatically not as extensions but as ad hoc types.

In our model we distinguish between object types, the instances of which must be maintained as separate sets, which we call *collection objects* and object types that have no instances and we refer to as *type objects*. Furthermore for efficiency and logical organization of collection objects, groups of predicate derived groups of instances are supported. These groups are called *aggregations*. Aggregations are rules, as will be shown in section 3.

We have adopted multiple inheritance which has been proved advantageous compared to simple inheritance because it simplifies data modeling. Two types of inheritance are supported. First, inheritance that indicates specialization, the so called ISA [Mylo80], and second, partial inheritance [Stro86], where some properties are inherited and others are suppressed.

As already mentioned the degree to which knowledge representation in an O-O DBMS is declarative or procedural is an important one. O-O database systems tend to be procedural because of concepts such as methods. Methods describe the behaviour of objects and due to their procedural nature, object

CH2728-4/90/0000/0556$01.00 © 1990 IEEE

functionality has no clear description. Furthermore, following the procedural approach, the need to express explicit constraints, leads to a mess.

Having taken all that into account we have adopted rules to express functionality and constraints. We have chosen a basic set of methods that a database object should have, and each such method we have attached to a rule with the corresponding conditions that must be satisfied so that the method preserves the integrity of the data it manipulates. Apart from rules involving basic operations, new rules can be defined by the programmer incorporating existing or user-defined methods.

As already mentioned, associations or relationships between objects are expressed by rules which are included into the behaviour part of objects. More specifically relationships are viewed as binary, bilateral associations between objects following the example of the binary data model. We call these relationships *disciplined relationships*. A disciplined relationship is structural knowledge common to a pair of objects, represented by a rule in each participating object.

So for two objects to be involved in a relationship, two rules with the same name must be defined, one for each object description. This may appear as rather unnatural but if the complete definition of a relationship were to be placed in both objects it would mean that constraints for both objects should be defined in one rule. That would compromise information hiding because the involved objects should know each other. As far as the user is concerned relationships are primarily meant to express a bond between two objects, so it is not important to have the full constraint at one place since constraints are automatically monitored by the system. Furthermore it is enough for an object, as a viewing mechanism, to contain information regarding its relationships with constraints that involve anything defined at that object.

The well known IS-PART-OF relationship expresses, in a declarative manner, that every instance of an aggregated object has associated with it instances of its components. In other words it expresses two things between an object and its components, firstly taxonomy through hierarchy and secondly reference between instances. Reference between an instance of an object and the instances of its components is much more a relationship than a mere constraint through inheritance rules since it must be clearly specified at any moment which instances are components of which instance. This type of special behaviour of the IS-PART-OF can be better expressed in a declarative form as a disciplined relationship which is referred to as the *PART-OF* disciplined relationship.

The complete representation of the IS-PART-OF relationship comprises the disciplined relationship PART-OF, which expresses association between the whole and its components, along with a TYPE-OF hierarchy, which expresses taxonomy. An important advantage of this representation of IS-PART-OF is that it supports both physical part hierarchy, in which an object cannot be part of more than one object, and logical part hierarchy, where one object can be part of more than one objects.

PART-OF can be used alone to define complex objects in two cases that cannot be covered by the standard approach for complex objects. The first case is when we want to build complex objects bottom-up, that is, to assemble existing objects in order to create a complex object. The second case is when we want to keep the component objects after the deletion of the parent object.

A database schema is a self-sufficient module that can serve as a component for modular design. That is, an application can be partitioned into interacting, logically independent parts and then each part can be independently represented by a schema. These separate schemas can then be interconnected via disciplined relationships to form the *environment* of the application. In such an environment the tight hierarchical organization of objects is not followed between schemas but instead the more loose disciplined relationships approach is adopted for flexibility without loss of consistency. This modular design approach, ie. with loosely coupled modules, enables the formation of distributed environments in a variety of configurations.

Disciplined relationships can be also used to interconnect distinct databases in a similar way as they serve to interconnect different schemas of the same database. Furthermore heterogeneous databases can be connected in the same way by transforming the corresponding data models to the one we propose.

From the discussion so far we can deduce that the basic idea of our Data Model is objects having "knowledge" of their relationships with the rest of the objects. This knowledge is private for each object. The idea of retrieval and data manipulation is based on these relationships and the object structure. A well accepted type of language for solving problems involving relationships between objects, is a logical one. More specifically, logic has provided a basis for relational databases and database theory, especially for expressing queries and defining views.

In view of the above we have chosen a formalism based on Horn Clauses. So from the data manipulation point of view, we have a logic database where the extensional database consists of objects that have attributes and rules. Furthermore the terms of queries always refer to specific objects and not to the whole data base.

The well defined semantics and the simple, well understood notation of Horn Clauses combined with the concept of disciplined relationships led us to a method for the evolution of database schemas in terms of new object interrelations. This method is based on transaction monitoring. Commonly used, simple query patterns appearing in queries are identified and mapped onto the database schema and onto the data. The advantage of this approach as opposed to saving specific queries is that query patterns provide an abstraction over a number of queries and are thus more useful in query optimization and more appropriate to be included into the database schema.

REFERENCES

[Ball88] N. Ballou, H. Chou, et al."Coupling an Expert System Shell with an Object-Oriented Database System", The Journal of Object-Oriented Programming, June/July 1988, pp. 12-21.

[Chri89] D. Christodoulakis, P. Soupos, S. Goutas, "Adaptive DB Schema Evolution via Constrained Relationships", proc. of the IEEE Intern. Workshop on Tools for Artificial Intell., George Mason Univ., Fairfax, Virginia, Oct., 1989.

[Gout89] S. Goutas, P. Soupos, D. Christodoulakis, "A New Approach Towards an Object-Oriented Database System", Proc. EUROMICRO 89, Cologne, Sept., 1989.

[Kim89] "Object-Oriented Concepts, Databases and Applications", W. Kim and F. Lochovsky (Eds.), ACM Press, New York, 1989.

[Mylo80] J. Mylopoulos, P. Bernstein, and H.K.T. Wong, "A language facility for designing database-intensive applications", ACM Trans. on Database Syst., vol. 5, no. 2, Jun. 1980, pp. 185-207.

[OOPS87] "Report on the Object Oriented Database Workshop: Semantic Aspects", Addendum to the Proc. of the OOPSLA '87 Conf., Orlando, Florida, 1987, pp. 45-50.

[Stro86] B. Stroustrup, "The C++ Programming Language", Addison - Welsey, Reading, Ma., 1986.