Object Modeling of RDBMS Based Applications

Giuseppe Polese
Università di Salerno, Italy

Vincenzo Deufemia
Università di Salerno, Italy

Gennaro Costagliola
Università di Salerno, Italy

Genny Tortora
Università di Salerno, Italy

INTRODUCTION

As the development of industrial software applications become more complex, the benefits of developing a comprehensive “blueprint” enabling developers to visualize the scope of a project increase substantially.

One of the key points in the development of most industrial applications is the design of the database. The object-oriented design paradigm enhances database modeling because the object model is expressive, concise, and relatively easy to use. On the other hand, the relational data model offers the advantages of a standard model, having a rigorous theoretical foundation, a mature technology, and the possibility to easily work with declarative languages. The complementary strengths of object and relational models makes it desirable to have them coexist in some form, achieving the best from both their paradigms (Li & Zhao, 2003; Musto et al., 2000).

The Object Modeling of Relational Applications (OMAR) methodology uses object-oriented front end modeling and supports the generation of relational database schemes for the designed applications (Musto et al., 2000). It uses inference mechanisms exploiting the data structuring information and the dynamic models produced during the object-oriented modeling phases to infer an appropriate relational data schema and the associated manipulation mechanisms. Thus, a designer still benefits from the advantages offered by the object-oriented modeling paradigm, without loosing the above mentioned advantages offered by the relational data model.

In this article, we show how we have used visual language technology to develop OMAR front-end CASE tools. A visual compiler generator, VLCC, has been used to model the syntax of UML diagrams and to generate a syntax driver editor and a compiler for them. Moreover, the inference mechanisms of OMAR have been embedded within VLCC by means of semantic routines. These are invoked upon the successful parsing of UML diagrams and produce an object/relational source code version of the application under development. We describe such a version of the code by using the PC++ language (Musto et al., 2000), a high level C++ providing abstract mechanisms to manage the communication with RDBMSs.

BACKGROUND

Applications that combine an object-oriented development method with RDBMS-based implementation can benefit from the use of a powerful, flexible, and expressive design paradigm and a standard implementation platform. In this way, application experts can focus on design instead of low-level tasks like tuning a database, just as high-level languages free programmers from repetitive tasks like assigning variables to registers. Any architecture that preserves this separation of concerns must have the ability to hide low-level details from the application expert. One way to do this is to automate mapping rules through an object-model compiler, having an RDBMS as its target architecture. Experts can then focus on the nuances of the application and avoid rediscovering database design principles for each new project. Moreover, the accuracy of the tables no longer depends on the person doing the conversion. One of the most important approaches for mapping object schemes onto RDBMS schemes is presented in Blaha, Premerlani, and Shen (1991). This approach combines the Object Modeling Tool (OMT) with Schemer, a compiler developed at GE Corporate R&D. In particular, Schemer converts the object schemes into SQL code, which can then be used to generate relational tables.
THE DEVELOPMENT PROCESS WITH OMAR

Nowadays, there is still high demand of compatibility between object-oriented development methodologies and traditional DBMS platforms. In fact, in spite of reduced modeling capabilities, traditional RDBMSs offer the advantages of a widely used technology that also allows to easily recruit or train experienced professionals in the field. On the other hand, the object-oriented paradigm has the potential to improve many aspects of the software development process, including a more effective maintenance and reuse of existing applications.

The OMAR methodology supports the development of RDBMS-based applications starting from object-oriented specifications or programs. It allows a designer to build large systems by using object-oriented design methodologies and to generate a prototype of the whole relational application with the support of automated tools. The OMAR development life cycle is depicted in Figure 1.

The PC++ programming environment relies upon the PC++ compiler, which extends the C++ language with abstract mechanisms to cope with persistent classes and objects to be stored and manipulated through relational DBMSs. The programmer uses high-level constructs to declare persistent classes and the object-oriented paradigm to interact with the stored data. It will be the compiler responsibility to generate the appropriate C++/SQL code using ODBC calls (Gryphon et al., 2000). The PC++ programming environment can be used as an entry point of the OMAR methodology for developing small relational applications, or it can be used as a back end for large applications modeled through the object-oriented design paradigm of OMAR. The object-oriented design schemes can be translated either in C++/SQL or PC++ code. The latter can in turn be translated in C++/SQL using the PC++ compiler.

MAPPING CLASS DIAGRAMS ONTO RELATIONAL APPLICATIONS

In the following, we provide algorithms to capture and process the information contained within class diagrams, state diagrams, and other behavior diagrams of an object design schema to derive an appropriate structuring of a relational data schema for the application under development and the associated manipulation code. In particular, the class diagram provides the necessary information to derive the tables to be generated. The remaining diagrams can be exploited to analyze the dynamic behavior of the system, which provides heuristics on how to refine the relational data schema to accommodate the data navigation requirements of transactions, and it is used to derive the manipulation code.

Extracting a Relational Data Schema from the Class Diagram

The rules for extracting a relational data schema from the class diagram of an application are an extension of the rules for mapping Extended Entity Relationship diagrams onto relational data schemes (Elmasri & Navathe, 2003). A sample of these rules is the mapping of Generalization/Specialization hierarchies.

A generalization/specialization hierarchy can be of different types depending upon two properties: disjunction/overlapping and partial/total (Elmasri & Navathe, 2003). Their combination yields the following four types of hierarchies: (1) disjoint/total, (2) disjoint/partial, (3) overlapping/total, and (4) overlapping/partial. If a generalization/specialization hierarchy of type 2, 3, or 4 has superclass C with attributes \{a_1, a_2, ..., a_n\} and m subclasses \{S_1, S_2, ..., S_m\}, then it is mapped by using the following rules:

- Create a table \(T_c\) with columns \(\{a_1, a_2, ..., a_n\} \cup \{OID_c\}\) corresponding to the attributes of class C plus its unique identifier (OID). The latter is used to manage inter-object references and will be a candidate primary key for the table \(T_c\).
- For each subclass \(S_i\), 0 < i < m + 1, create a table \(T_{c_i}\) with columns \{attributes of \(S_i\)\} \(\cup \{OIDS_{S_i}\}\). Once again, here the OID will be a candidate primary key for the table \(T_{c_i}\).
- Create a system table INHERITANCE, if it does not exist. This table will be updated with the insertion of