Converting a Legacy Database to Object-Oriented Database

Reda Alhajj  
*University of Calgary, Canada*

Faruk Polat  
*Middle East Technical University, Turkey*

**INTRODUCTION**

We present an approach to transfer content of an existing conventional relational database to a corresponding existing object-oriented database. The major motivation is having organizations with two generations of information systems; the first is based on the relational model, and the second is based on the object-oriented model. This has several drawbacks. First, it is impossible to get unified global reports that involve information from the two databases without providing a wrapper that facilitates accessing one of the databases within the realm of the other. Second, organizations should keep professional staff familiar with the system. Finally, most of the people familiar with the conventional relational technology are willing to learn and move to the emerging object-oriented technology. Therefore, one appropriate solution is to transfer content of conventional relational databases into object-oriented databases; the latter are extensible by nature, hence, are more flexible to maintain. However, it is very difficult to extend and maintain a conventional relational database.

We initiated this study based on our previous research on object-oriented databases (Alhajj & Arkun, 1993; Alhajj & Elnagar, 1998; Alhajj & Polat, 1994, 1998); we also benefit from our research on database re-engineering (Alhajj, 1999; Alhajj & Polat, 1999). We developed a system and implemented a first prototype that takes characteristics of the two existing schemas as input and performs the required data transfer. For each of the two schemas, some minimum characteristics must be known to the system in order to be able to perform the transfer from tuples of the relational database into objects in the object-oriented database. We assume that all relations are in the third normal form, and the two schemas are consistent; that is, for every class, there is a corresponding relation. This consistency constraint necessitates that attributes with primitive domains in a certain class have their equivalent attributes in the corresponding relation, and values of non-primitive attributes in a class are determined based on values of the corresponding foreign keys in the corresponding relation. Concerning the migrated data, it is necessary that consistency and integrity are maintained. Finally, the transfer process should not result in any information loss.

**BACKGROUND**

There are several approaches described in the literature to facilitate accessing content of a relational database from an object-oriented application. DRIVER (Lebastard, 1995) proposes an object wrapper that allows relational database reusing. It uses relational database management systems as intelligent file managers and proposes object models on top of them. The user is expected to provide the mapping between the two schemas. Persistence (Agarwal, Keene & Keller, 1995; Keller, Agarwal & Jensen, 1993) is an application development tool that uses an automatic code generator to merge C++ applications with relational databases. The application object model is mapped to a relational schema in the underlying database. Therefore, object operations are transformed into relational operations and vice versa. The benefits and risks of the migration process are discussed in Keller and Turner (1995). The authors argue that storing data in a relational database and providing a wrapper to allow programming in an object programming language provides more benefit at significantly reduced risks and costs as compared with migrating to an object-oriented database. We argue that providing a wrapper adds a performance cost to be paid every time the data is processed because the mapping between objects and tuples is repeated dynamically in each session. But, the mapping cost is paid only once by migrating to an object-oriented database.

**MAIN THRUST**

In this section, we present the basic necessary analysis that results in the information required to transfer the
Converting a Legacy Database to Object-Oriented Database

data safely. First, we identify characteristics of the object-oriented schema. Then, we concentrate on characteristics of the relational schema. The last step leads to the equivalence between attributes in the two schemas. This information directs the system about the construction of objects out of tuples.

We investigate characteristics of the given object-oriented schema and construct the necessary tables. First, we present the basic terminology and definitions required to understand the analysis. We are mainly interested in the following class characteristics:

1. \( C_p(c) \): list of the direct superclasses of class \( c \).
2. \( L_{attributes}(c) \): set of (not inherited) attributes locally defined in class \( c \).
3. \( L_{instances}(c) \): set of identifiers of objects added locally to class \( c \).
4. \( W_{instances}(c) \): extent of class \( c \), that is, objects in \( L_{instances}(c) \) and in all direct and indirect subclasses of class \( c \).
5. \( OIDG \): identifier generator that holds the identifier to be granted to the next object to be added to \( L_{instances}(c) \).

Definition 1

[Domain] Let \( c_1, c_2, \ldots, \) and \( c_n \) be primitive and user-defined classes, where primitive classes include reals, integers, strings, and so forth. The following are possible domains:

1. \( (a_1:c_1, a_2:c_2, \ldots, a_n:c_n) \) is a tuple domain; a possible value is a tuple that consists of object identifiers selected from classes \( c_1, c_2, \ldots, \) and \( c_n \), respectively.
2. \( c_1, 1 \leq i \leq n \) is a domain; a possible value is an object identifier from class \( c_i \).
3. \( d \) is a domain, which may be any of the two domains defined in 1 and 2; a possible value is a set of values from domain \( d \).
4. \( [d] \) is a domain, which may be any of the two domains defined in 1 and 2; a possible value is a list of values from domain \( d \).

Example 1: [Object-Oriented Schema]

Person:
\( C_p(Person)=[] \)
\( L_{attributes}(Person)=\{SSN:integer, name:string, age:integer, sex:character, spouse:Person, nation:Country\} \)

Country:
\( C_p(Country)=[] \)
\( L_{attributes}(Country)=\{Name:string, area:integer, population:integer\} \)

Student:
\( C_p(Student)=[Person] \)
\( L_{attributes}(Student)=\{StudentID:integer, gpa:real, student_n:Department, Takes:\{(course:Course, grade:string)\}\} \)

Staff:
\( C_p(Staff)=[Person] \)
\( L_{attributes}(Staff)=\{StaffID:integer, salary:integer, works_in:Department\} \)

ResearchAssistant:
\( C_p(ResearchAssistant)=[Student, Staff] \)
\( L_{attributes}(ResearchAssistant)=\{} \)

Course:
\( C_p(Course)=[] \)
\( L_{attributes}(Course)=\{Code:integer, title:string, credits:integer, Prerequisite:\{Course\}\} \)

Department:
\( C_p(Department)=[] \)
\( L_{attributes}(Department)=\{Name:string, head:Staff\} \)

Secretary:
\( C_p(Secretary)=[Person] \)
\( L_{attributes}(Secretary)=\{words/minute:integer, works_in:Department\} \)

Related to the object-oriented schema, the analysis is based on the domain information summarized in:

ObjectAttributes(class name, attribute name, domain, Score)

Table 1. ObjectAttributes: A list of all attributes with non-primitive domains

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Attribute Name</th>
<th>Domain</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>spouse</td>
<td>Person</td>
<td>1</td>
</tr>
<tr>
<td>Person</td>
<td>nation</td>
<td>Country</td>
<td>1</td>
</tr>
<tr>
<td>Student</td>
<td>student_in</td>
<td>Department</td>
<td>1</td>
</tr>
<tr>
<td>Student</td>
<td>Takes</td>
<td>Ti</td>
<td>2</td>
</tr>
<tr>
<td>Staff</td>
<td>works_in</td>
<td>Department</td>
<td>1</td>
</tr>
<tr>
<td>Course</td>
<td>prerequisite</td>
<td>Course</td>
<td>2</td>
</tr>
<tr>
<td>Department</td>
<td>head</td>
<td>Staff</td>
<td>1</td>
</tr>
<tr>
<td>Secretary</td>
<td>works_in</td>
<td>Department</td>
<td>1</td>
</tr>
<tr>
<td>Ti</td>
<td>course</td>
<td>Course</td>
<td>1</td>
</tr>
</tbody>
</table>