Translating OODB method to RDB routine

Joseph Fong
Department of Computer Science, City University of Hong Kong
Tat Chee Avenue, Hong Kong, email: csjfong@cityu.edu.hk

San Kuen Cheung
Department of Computer Science, City University of Hong Kong
Tat Chee Avenue, Hong Kong, email: cheungsk@asiaonline.cityu.edu.hk

Abstract

A methodology is introduced for translating from methods in an object-oriented database to routines in a relational database. The approach consists of three steps. The first step is to translate method signature to Persistent Stored Modules signature. The second step is to translate method source language to (function/procedure) routine. The process includes Host Language, OSQL's Qualification, Query Translation, Update Transaction Translation, and Objects inside Object. The third step is to translate method invocation to routine invocation. According to this approach, object-oriented database methods can be translated to relational database routines that can be executed in the relational database environment assisted by a frame model and case statements listing all possible cases of binding conditions and actions. The significance of the finding is adding an open object-oriented interface on top of relational database system for database interoperability and in the development of an object-relational database management system.

1 INTRODUCTION

1.1 Background

In the current situation, many companies are using the relational concepts and products for assisting their business decision. Problems in using OODB are in the reliability, performance, cost of migration, and lack of expertise. Therefore, a legacy system installed the object notion is our motivation. Some research papers have introduced some related works regarding the migration from OODB to RDB and vice versa. We all know that the current market has many ORDBMS (Object-Relational Database Management Systems). However, they did not consider migrating object method to its relational counterpart. In this paper, we shall translate from object methods to relational routines. After implementing our methodology in a RDB, users can apply object features in their relational database according to our concepts.

1.2 Architecture of Method Translation

Figure 1 shows that the mapping process consists of three steps. Step 1 is Schema Translation in where method signature will be mapped to PSM signature and stored in a frame model. Mapping the signature is a predefined process and is operated in the Schema Translation. Step 2 is Method Source Translation in where object methods will be converted to relational routines. Step 3 is Invocation in where dynamic binding of an object feature can be applied in relational database.

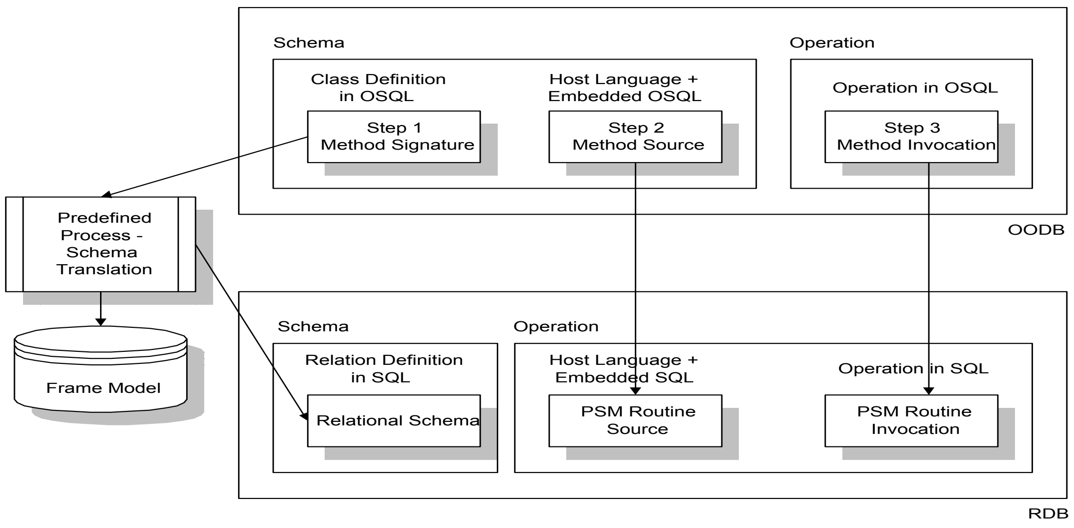

Figure 1 – Mapping from Object Method to PSM Routine

1.3 Motivation and Contribution of the Work

We are interested by the object technology so that we want to expand features of the RDBMS so as to handle object facilities. The contribution of our research is on the schema level. Translating the dynamic schema, especially on definition and behavior of the object method, from OODB to RDB plays an important role on an ORDBMS. Adding a feature of routines in the relational database can increase its functions like the methods in the object-oriented database. The migration of the object feature in the RDB is for building up an ORDBMS in which object and relational features can come into a solution for the user requirements. Before going to create the ORDBMS, we must solve some conflicts between object-oriented schema and relational schema.

By adding up the object technology, a relational database can be increased in the reusability and portability. In our further research, we want to create an object-relational database management system (ORDBMS) by creating a virtual interface of the object-oriented model in relational system. The existing data of the relational model can be reused without interrupting daily operations of the existing system. After implementing the interface, companies can realistically utilize the benefits of object-relational database system.

1.4 Overview of the Architecture

The approach consists of meta-data (i.e. Frame Model), method translation, and relational database engine. The architecture of our approach is outlined in Figures 1 and 2. A relational schema and data of frame model will be generated in the process of the schema translation. All the table definitions will be defined in the relational schema according to the schema of object-oriented model. The frame model contains four system classes which are called header, attribute, method, and constraint. The core of the frame model is in these classes. Afterwards, the interoperability between the object message and relational database engine will be relied on the frame model.

Method translation is the most difficult part of the database re-engineering. A methodology is introduced for translating from methods in an object-oriented database to routines (or stored procedures) in a relational database. The approach consists of three steps. The first step is to translate method signature to Persistent Stored Modules signature. The second step is to translate source language of methods to target language of routines (consisting either functions or procedures). The process includes Host Language, OSQL's Qualification, Query Translation, Update Transaction Translation and so on. The third step is to translate method invocation to routine invocation. According to this approach, methods can be translated to routines that can be executed in the relational database environment assisted by a frame model and case statements which list all possible cases of binding conditions and actions.

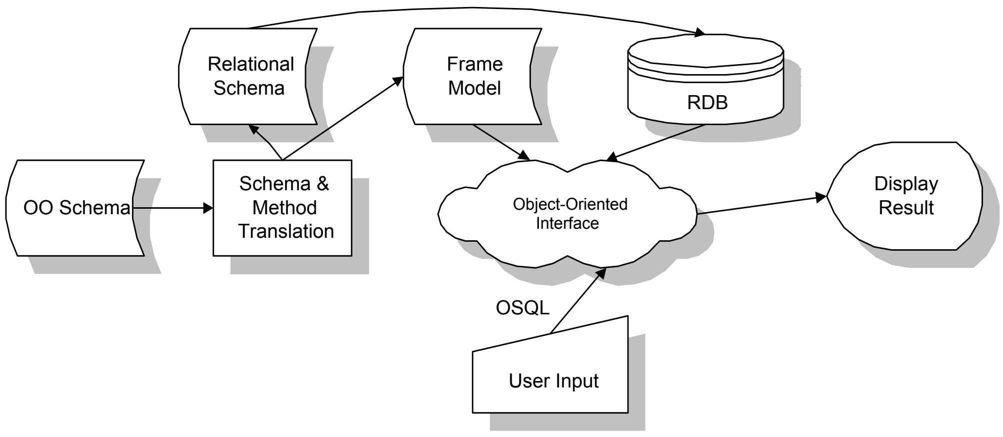

Figure 2 – Architecture of ORDBMS

2 RELATED WORK

We introduce a methodology to translate the object methods to RDB routines. In this area, some research papers have already introduced related works such as frame model, schema translation, data type mapping, and object-relational database.

2.1 Frame Model

Fong & Huang [5] translated existing data models into a frame model of the universal database. The structure of frame model consisted of several classes such as Header, Attributes, Methods, and Constraints classes. According to the frame model, a universal database could be formed. As a result, old and new database systems could be coexisted to form a data warehouse for a decision support system.

2.2 Schema Translation

Blaha et al., [6] discussed converting OO Models into RDBMS schema. The approach combined OMTool with Schemer. The Schemer converted the object model into SQL code, which could be used to generate relational tables. Firstly, it read the logical model in the ASCII file produced by OMTool, and populated the object meta-model. Secondly, it mapped object-oriented constructs from the meta-model into ideal RDBMS tables using the ideal-table meta-model. Thirdly, it mapped the ideal tables to the SQL dialect supported by the target RDBMS.

2.3 Data Type Mapping

DeFazio & Srinivasan [14] extended RDBMS for handling complex domains. Their approach required (1) modeling complex types with user defined types, (2) delivering domain-specific behavior using a combination of methods and client-based software, and (3) implementing domain-specific indexing and retrieval methods. The approach required applications to integrate layered products from independent software vendors (ISVs) to deliver specific functionality for a domain. P. Seshadri [20] argued that the next generation of object-relational database systems should be based on enhanced abstract data type (E-ADT). Several E-ADTs have been developed for complex types like images, audio, video, raster, polygons, etc. The ORDBMS maintains a table of ADTs, and new ADTs may be added by database developers. Each ADT can also declare primitive methods for manipulating or querying values of the type. Libraries of primitive methods for each ADT are sometimes called “datablades” (Informix), “data extenders” (IBM), or “data cartridges” (Oracle).

2.4 Object-Relational Database

In [32], Oracle server can be enhanced by developers to create their own application-domain-specific data types so as to bring the Object-Relational technology to the mainstream. The paper introduced a server-based components called Data Cartridges to integrate these new domain types as closely as possible with the server so that they could be treated at par with the built-in types like Number or Varchar. Informix [33] a collection of data types, their associated functions and operators, and access methods into DataBlade modules, using the metaphor that the DBMS is a razor into which DataBlade modules are inserted. An Object-Relational Database can be formed using the DataBlade modules. Software AG's Bolero provides an enterprise-wide middleware called EntireX for integrating new generation application components with the existing database systems. Users can apply Object-Oriented applications in a relational database DBMS.

3 META-DATA

3.1 Frame Model

The frame model follows the object-oriented paradigm and acts as meta-classes [25]. All conceptual entities are modeled as objects. The same attribute and behavior objects are classified into an object type, called a class. An object belongs to one, and only one class. Both facts and actions are objects in the frame model. The frame model is implemented with a knowledge representation schema that includes object structure classes, user-defined relationships between entities, and structure inheritance classes defined by taxonomies of structure that support data and behavior inheritance as shown in the following list. We employ the Frame Model because it has been established in 1997 [5]. The model can easily be modified according to the different situations. It can be formed a meta-data during the schema translation and can be constructed for database interoperability. Also, it allows designers to combine the rules together into a class and to associate the classes as a knowledge-based system.

3.2 Frame Structure

During the schema translation, the proposed translator will generate two semantics of a frame model: static and dynamic semantics. The static semantic consists of Header and Attribute classes and the dynamic semantic consists of Method and Constraint classes. The Header class includes basic object information to represent the class entity. The Attribute class represents the properties of classes. The Method class includes actions to extend the semantics of the data. The Constraints class consists of properties of data that cannot be captured in the form of structure [5].

Header Class

```
{   Class_name          // a unique name in all system
    Parents             // a list of superclass names
    Operation         // program call for operations
    Class_type          // active or static class }
```

Attribute Class

```
{   Attributes_name     // an attribute in this class
    Class_name          // reference to header class
    Method_name         // a method in this class
    Attributes_type     // attribute data type
    Default_value       // predefined value
    Cardinality         // single or multi-valued
    Description         // description of the attributes }
```

Method Class

```
{   Method_name         // a method component in this class
    Class_name          // reference to header class
    Parameters          // No. of arguments for the method
    Method_type         // return type of method
    Condition       // the rule conditions
    Action              // the rule actions }
```

Constraint Class

```
{   Constraint_name     // a constraint component of this class
    Class_name          // reference the header class
    Method_name         // construct method name
    Parameters          // No. of argument for the method
    Ownership           // the class name of method owner
    Event               // triggered event : create update or delete
    Sequence            // method action time : before or after
    Timing              // the method action timer }
```

4 METHOD SIGNATURE TO PSM SIGNATURE (SCHEMA TRANSLATION)

Mapping the name of a method requires avoiding the potential of name conflicts because a PSM routine is not bound by a specific relation (i.e. table), whereas an OODB method is bound by a class. It is necessary to rename the mapped PSM routines uniformly in the RDB system. One way is to name a mapped PSM routine by combining the class name and the method name and assign an "action" name to the routine as well as to the meta-data.

Mapping the parameters listed in a method requires each data type of parameter related to a class to be converted to a corresponding data type related to relation that spans the same data content as the class. This mapping can be determined by observing the data type mapping from classes to relations. Mapping the return types is the same as mapping the parameters. The return types of methods are stored in the “method” class as a meta-data. The relational types of return values are defined by mapping the object data types and the relational data types. Inside the meta-data, the return types are the relational preferences.

4.1 Pseudo Code for (Schema Translation)

Algorithm :

```
program schematran              // This is for reading superclass(es).
OPEN OODB schema
WHILE not at end of classes in OODB schema DO
begin   //This is for getting the information of superclass(es).
        get a class;
        IF class has superclass(es)
        THEN save class and superclass(es) in memory;
end;
WHILE not at end of classes in OODB schema DO
// This is for extending inheritance and building oid field for each table
begin
        get a class;
        save the class into a table and header class of frame model;
        insert attribute oid value into the table;
        IF the table has superclass // the superclasses must be in front of the
        THEN begin   // subclasses on the list of OODB schema. So, their attributes
            get superclass’s attributes and methods; // will be read before their
            save attributes to relational schema and   // subclasses.
                the attribute class of frame model;
            save methods into the method class of frame model;
        end;                                      // **signature** to frame
        get own attributes; //This is for re-organizing attributes and atomizing
        IF data type of attribute = multiple-valued     // aggregate attributes
        THEN begin
            map OODB type to RDB type;      // changing data type
            copy oid and multiple-valued attributes to relational schema of a new table;
            save the new table to the header class of frame model;
            save the attributes of new table to the attribute class of frame model;
        end;
        IF data type of attribute = OID // This is a user-defined data type
        THEN begin
            map OODB type to RDB type;      //changing data type
            copy oid attribute as a foreign key of an association table to R schema;
        end;
        IF data type of attribute = preliminary data type
        THEN save own class attributes to relational schema and attribute class of frame;
        IF data type of attribute = method
        THEN save own methods to method class of frame model;
        end;                                // signature to frame model
end;
```

We must change the data types of the object-oriented model for the target model (i.e. relational model); but they depend upon specific product of the relational model. Different products provide different data types. During our research, we found that each product had defined its own data types.

5 METHOD TO PSM ROUTINE

5.1 Overview of the Method and Routine

Before introducing the methodology, we must clarify what are the important parts we are going to consider in our approach. In the object model, method definition is defined in object schema. In the relational model, the routine definition is not a part of relational schema. During the schema translation, Method can be considered as an attribute in which the value is not statically stored in database, but the body is dynamically calculated by executing from a related program. The method signature must be identified and stored in data dictionary. A combination of method name and its parameters within a class can be identified a specified method uniquely. This is a feature of object model. In relational schema, there is no such a mechanism that supports user-defined functions and procedures. Each routine name must be identified a routine uniquely. Commercial SQL products have been offering such a capability for years [12] in the form of stored procedures and the SQL standard has also begun considering to include this capability of PSM routine [11, 12], in both user-defined functions and procedures.

In the object model, source code of methods can be defined by object definition language (ODL) in the schema level. A method source can be constructed by combining embedded OSQL and host language (e.g. C or C++). In the relational model, source code of routines is defined in the operational level (see Figure 3). A routine source is constructed by combining embedded SQL and host language. In this situation, we do the method translation.

Methods can be operated by OSQL in which a method can be invoked by an object feature of navigation. Routines contain functions and procedures. An user-defined function is invoked by scalar expressions. An user-defined procedure is invoked by a new SQL statement (typically “do” or “call”). During the method translation, all the object methods will be translated to relational routines/procedures. Routine invocation is assisted by case statement which lists all possible cases of binding procedures. The method/routine invocation is the finally step of our approach (see Figure 1).

5.2 Method Source Translation

The statements of embedded OSQL must be mapped to the embedded SQL statements. The statements of host programming language which deal with the flow control and operations on volatile data do not need to change (assuming that host languages are the same for both OODBMS and RDBMS, otherwise only key words and statement formats must be adjusted). The statements of host programming language which deal with operations on non-volatile data must be modified according to data type mapping table from classes to relations. Mapping the body of a method requires distinguishing the above statements and different statements may employ different mapping rules.

Rule 1 : Path expression operand translation
Rule 2 : Set operand translation
Rule 3 : Query translation
Rule 4 : Update transaction translation
Rule 5 : Host language data type translation (for non-volatile data)
Rule 6 : Objects inside object translation

An object method consists of embedded OSQL statements and host language. The main problem of the method translation is on the embedded OSQL statements. There are many issues such as qualification translation, query translation [10], and update transaction translation. We use path elements to handle the qualification translation (i.e. where-clause). The idea of the path elements is that we shall locate the path expression of an attribute or a method. The meta-data contains all the information of each attribute or method from the schema translation. The path elements can be located from this meta-data. After forming a path logic of a specific attribute or method, we can evaluate all the elements in the path logic. The path logic in the where-clause can be translated to a join operation in the RDB system. The qualification translation is not only on the path expression operand, but also on the set operand.

The set operand in the object model can be treated as an object feature of multiple-valued attribute. The feature provides a searching algorithm for the multiple values. Our resolution for the multiple-valued attribute in the relational schema is to create a new individual table in which each value of the multiple-valued attribute can be stored in each individual tuple. The location of this attribute can be obtained from the semantic formed by the meta-data.

We must consider the method compiler from two systems (object and relational engine). A common host language such as C or C++ can be accepted by both systems. The host language may not be necessary to convert. Variables of the host language which deal with operations on nonvolatile data must be considered according to data type mapping table from classes to relations. During the schema translation, classes of the object model are converted to tables of the relational schema. The object definition and class attributes will no longer exist on the relational database. In this case, we must extract these variables from the object method and convert the data structures of these variables for operations on the relational database.

5.2.1 Rule 1 - Path Expression Operand Translation

In path expression, address of a method or an attribute is an important part of the statement. The translator will go through the path logic for locating the method or attribute and will execute the method or retrieve the value of the attribute. We must know how to interpret the path logic of an object-oriented navigation and to reconstruct the logic to a relational statement. The first step is to decompose the object path which consists of composite attributes or other user-defined methods. The composite attribute will return its value to operator in comparison with other value. The method will execute its calculation and return the result to operator for doing the same comparison as the composite attribute. The navigation of a class is decomposed during the schema translation. Important information is stored on the meta-data and on the static tables. The second step is to organize the path logic of the object-oriented navigation for relational statement. Since the information or detail of the class has been stored in a repository as meta-data during schema translation, we can create the path logic of the relational statement from extracting the information in the meta-data.

Algorithm :

```
program path_expression        // This is for decomposing path expression.
WHILE not at end of comparison operator's elements DO
// operators in "where-clause"
begin
    get elements from operator; // left or right side totaling 3 elements
                                // (e.g. A.ar.arr = 11)
    get components of each element; //each element has its components
                                     // (e.g. A.ar.arr)
```

```
        map components from header & attribute class of frame model;
       //collecting table name & association (i.e. A table and ar's table as a path logic).
        save table and its sub-ordinate tables to &tables;
        //for constructing join operation for SQL statement.
        get data type from attribute class of frame model; //e.g. attribute arr's data type
        CASE data type of
        "preliminary data type" :
              save attribute to &preliminary;
               IF attribute in "where-clause"
               THEN begin
                      put join operator(s) for &tables into SQL if necessary
                      put &preliminary into SQL for comparing with "11"
               end;
        "method parameter" :
              save the name of action attribute to &method;
               IF attribute in "where-clasue"
               THEN put &method and parameters into SQL;
        END-CASE;
end;
```

In this path expression translation, we want to analyze the composite attributes in the where-clasue. Actually, the translation can apply to other composite attributes within the OSQL statements. For instance, you can apply the Rule 1 to analyze the composite attributes in the "select-clause" in query statement and "set-clause" in update statement. Now, the where-clause has been constructed a true relational syntax according to the composite attribute of object-oriented navigation.

5.2.2 Rule 2 - Set Operand Translation

In this rule, we are still focusing to analyze a set attribute on the where-clause. The path logic of the set attribute involves a new individual table that contains the multiple values of a set attribute. We decompose the multiple values into a new table during the schema translation. The new table is created for containing the multiple values of each attribute. Primary key on the existing table and composite key on the new table link the connection between the existing table and the new table up. The composite key consists of two attributes. The following is an algorithm for the path logic of the set attribute.

Algorithm :

```
program set_operand     // This is for decomposing the set operand.
```

```
WHILE not at end of comparison operator's elements DO
// operator in "where-clause"
begin
    get elements from operator; // left and right side totaling 3 elements
    get components of each element; //each element has its components
    map components to header and attribute classes of frame model;
    //collecting table names & association between existing and new tables.
    get data type from attribute class of frame model;
    IF attribute data type = table name with prefix of "new" located in header class
    // i.e. set attribute by verifying against data type in attribute class
    THEN begin
        save table and sub-tables from attribute class of frame model to &tables;
        get attribute from attribute class of frame model;
            IF attribute type = preliminary data type       //.i.e. multiple-value attribute
            THEN save attribute to &preliminary;   // i.e. attribute from new table
                IF attribute in "where-clasue"
                THEN begin
                    put join operator(s) for &tables into SQL if necessary
                    put &preliminary into SQL
                end;
            end; end; end;
```

The &table and &preliminary are temporary variables that are the important part in constructing the path logic. The existing and new tables will be stored in the array named &tables. The name of the multiple-value attribute in the new table may be changed for internal identification. We use &preliminary as a temporary storage for this name. The decision of identifying an attribute that is a multiple-value attribute relies upon the prefix with "new" in the "data type" attribute of the "attribute" class of the frame model. The name will be confirmed by locating the name in the "header" class of the frame model. A true relational syntax can be formed on SQL statement according to the set operand of the object-oriented navigation.

5.2.3 Rule 3 - Query Translation

A query statement consists of range part (select-clause), target part (from-clause), and qualification part (where-clause). From the range part, we shall apply the rules 1 & 2 on composite and set attributes. After getting some relevant tables from the range part, these tables will be placed on the target part. As for the qualification part, we need to consider (i) tables to be involved, (ii) methods to be invoked, (iii) join operations to be executed. Table names on the target part are factor for forming join operations. Rules 1 & 2 are solutions for decomposing the navigation on the qualification part. Methods will execute their calculations and return results to operator before doing the comparisons with other values.

Algorithm :

```
program query_translation
// This is for translating OSQL query statement to SQL query statement.
WHILE not at end of attribute list DO // analyzing the “select-clause” of the OSQL
begin
     CASE attribute of
          “preliminary data type” : begin
               save table from attribute class of frame model to &tables;
               save attribute from attribute class of frame model to &preliminary;
            end;
          “multiple value” : begin
             call set_operand       //applying rule 2
             end;
          “composite value” : begin
             call path_expression  //applying rule 1
             end;
     END-CASE;
end;
put &tables.&preliminary into SQL;            //putting in “select-clause”
put &tables into SQL;                  //putting in “from-clause”
WHILE not at end of logical operator’s elements DO   // for “where-clause”
begin      //e.g. “A.ar = 11 and”, the logical operator = “and”
    save logical operator to &logical_operator;
    CASE attribute of
          “preliminary data type” : begin
               save table from attribute class of frame model to &tables;
               save attribute from attribute class of frame model to &preliminary;
               end;
          “multiple value” : begin call set_operand end;            //applying rule 2
          “composite value” : begin call path_expression end;    //applying rule 1
    END-CASE;
    put &tables.&preliminary into SQL
    put &logical_operator into SQL;    //putting in “where-clause”
end;
```

The idea of the query translation is that we take the whole statement for our consideration. Firstly, we extract the attributes in the “select-clause” one by one. Some relevant tables will be stored in a temporary variable named &tables. The attribute name will also be stored in a temporary variable named &preliminary. It will print in SQL’s “select-clause” later. Secondly, we can print all the tables involved from &tables to the “from-clause”. Finally, we analyze the qualification part (i.e. “where-clause”) that may be divided by several sectors. These sectors will be broken by logical operators. Some join operations may be formed according to how many path elements in the path logic.

5.2.4 Rule 4 - Update Transaction Translation

In the update transaction translation, the target part (update-clause), range part (set-clause), and qualification part (where-clause) compose an update transaction statement. In order to identify attributes of the range part, we apply rules 1 & 2. This part is to determine how many update SQL statements to be performed. If an attribute contains multiple values, we shall execute more than one update SQL statement. An attribute with single value is easy to update as we do not have to consider the existing value. We have to consider the existing values before going to update an attribute with multiple values. These values are stored in an individual table in which the key is a composite key. To update a tuple with multiple values, we must know the old value and OID value. The qualification part is similar to the query translation.

Algorithm :

```
program update_transaction   // This is for translating update statement.
WHILE not at end of comparison operator’s element DO //analyzing “set-clause”
begin
     save comparison operator to &comp_operator;
     get elements from the comparison operator;        //left or right side
     get components from each element;
    IF data type = multiple value
    THEN call set_operand;         //applying rule 2
    ELSE IF data type = path expression
    THEN call path_expression;   //applying rule 1
    ELSE IF data type = preliminary data type
    THEN begin
         save table from attribute class of frame model to &tables;
         save attribute from attribute class of frame model to &preliminary;
    end;
    put new value from OSQL to &new_value          //It can be a multiple-value attribute
    IF value = multiple value
    THEN begin
        get oid value and old value from RDB table;
        save oid value and old value to &oid_value and &old_value;
     end'
end;
WHILE not at end of logical operator's elements DO        // for "where-clause"
begin
    save logical operator to &logical_operator;
    IF "preliminary data type"
    THEN begin
         save table from attribute class of frame model to &tables;
         save attribute from attribute class of frame model to &preliminary;
     ELSE
         call path_expression and set_operand    //applying rules 1 and 2
     end;
end;
WHILE not at end value of &new_value DO
begin
    put &tables into SQL;     //putting in "update-clause"
    put &preliminary, &comp_operator, &new_value into SQL; //putting in "set-clause"
    put &oid_value and &old_value into SQL;          //putting in "where-clause"
end;
```

In OSQL's update statement, a multiple-value attribute can be updated by one statement. In SQL's update statement, firstly, we must know the cardinality of the attribute to be updated. For a multiple-value attribute, we analyze tables to be involved and attribute names to be changed respectively. The tables and attributes will temporarily be stored in variables named &tables and &preliminary. We save the new values to an array named &new_value. Secondly, we extract the old values and oid value from database and store these values in variables named &old_value and &oid_value. Thirdly, we check the "where-clause" that is the same as the query statement. Finally, we can form a SQL statement by grouping all the variables.

5.2.5 Rule 5 - Host Language Data Type Translation (for Non-volatile Data)

The problem is on the data type. We do not know which host variable will store data extracted from the relational database or compare with data from the relational database.

Thus, we shall convert all the data types of host variables to the relational model. When the data type of host language does not match with the data type of host variable, the target application will then help us to solve the problem. An error procedure will be started, when a dismatch message of the data type occurs. The error procedure consists of a mapping table and a conversion process. The function of the mapping table is to map the relational data type to corresponding object data type. The relational data will be converted to object data (see Figure 3). In this approach, we do not have to worry about the host variable which deals with nonvolatile data.

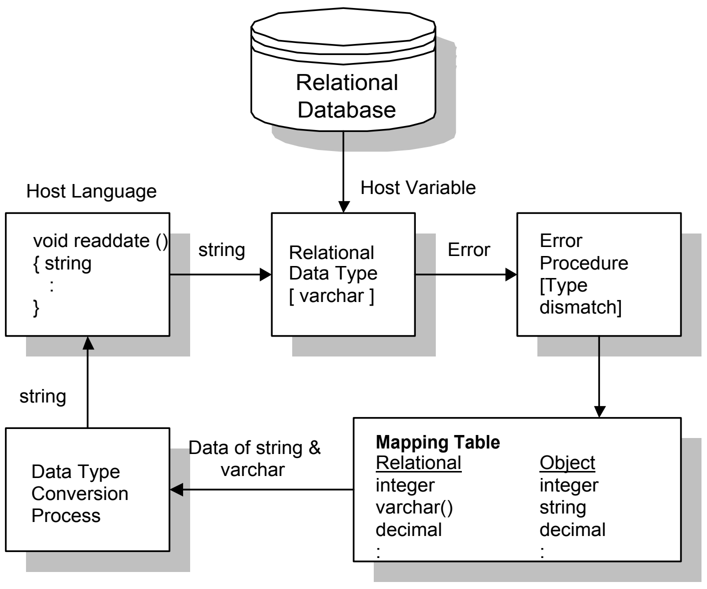

Figure 3 – Mapping Data Types

Algorithm :

```
program host_language   // This is for translating data type.
begin
    get value from database;
    get mapping table;
    map source data type to target data type;
    map target data to source data;
    copy the value to source;
    copy the source value to program;
end;
```

5.2.6 Rule 6 - Objects Inside Object Translation

An object can be a method. The method can invoke itself or other objects. We must convert the signatures of the object model to relational model so as to invoke the appropriate routines.

All the routine definitions are defined in the “method” class of the frame model instead of the relational schema. We collect the information of these methods from the case statement. We find the routine names from the “method” class of the frame model.

Algorithm :

```
program objects_inside   // This is for translating objects inside object.
begin
    get method name from OSQL;
    map the method name from case statement;
    map the method name from frame model;
    get the routine name from frame model;
    put the routine name into SQL;
end;
```

6 INVOCATION

6.1 Overview of the Invocation Translation

Enlarging the existing features of the relational schema is so important because the RDBMS cannot dynamically resolve the polymorphism. If we do not change the method name during the schema and method translations, the routine or stored procedure name will not be a unique key in SQL-92 because of the effect of the polymorphism. The routine names must be unique in the relational schema by their parameter list. We form an "action" attribute to distinguish routines between parent table and child table. It can be a code such as OID that can be stored in the "method" class of the frame model. Since the RDBMS does not have the dynamic binding, if we translate the method name directly to routine name without any changes, a specified routine may not be invoked by PSM's “call” or “do” statement. The “call” or “do” statement is for calling a procedure and the "select" statement is for calling a function.

6.2 Methodology of the Invocation Translation

The method invocation can be appeared in two different formats. The first one is like a procedure call, an invocation statement. The second one is like a function call, an operand in a scalar expression which is part of an OSQL statement. Mapping the method invocation must consider the polymorphism (dynamic binding) feature. It is not enough to map a method invocation to a corresponding PSM routine invocation because RDBMS can not dynamically resolve PSM routines in the same way that OODBMS can do for methods at run time. So it requires that mapped target for a method invocation should involve dynamic resolution of a PSM routine as well as invocation of the routine. One way is to use, in mapped target, case statement which lists all possible cases of binding conditions and calls of related PSM routines under each case. Polymorphism, Inheritance, and Multiple Inheritance are features of object model. We are going to discuss each possible case occurred in the relational model.

<u>Polymorphism</u>

Class definition

A (A1,…, M1 (a), M1 (a,b))

Relational definition

A (A1,…) with A_M1_1, A_M1_2

<u>Inheritance</u>

Class definition

A (A1,…, M1 (a), M1 (a,b))

B (B1,…, M1 (a)) subclass of A

C (C1,…, M1 (a,b,c) subclass of A

Relational definition

A (A1,…) with A_M1_1, A_M1_2

B (A1, B1,…) with B_M1_1, A_M1_2

C (A1, C1,…) with C_M1_1, A_M1_1, A_M1_2

<u>Multiple inheritance</u>

Class definition

A (A1,…, M1 (a), M1 (a,b))

B (B1,…, M1 (a)) subclass of A

C (C1,…, M1 (a,b,c) subclass of A, B

Relational definition

A (A1,…) with A_M1_1, A_M1_2

B (A1, B1,…) with B_M1_1, A_M1_2

C (A1, B1, C1,…) with C_M1_1, A_M1_1, A_M1_2

Meta-Data (Method class of the Frame Model)

Method name	**Class name**	**Parameters**	**Method type**	**Condition**	**Action**
M1	A	a	void	nil	A_M1_1
M1	A	a,b	void	nil	A_M1_2
M1	B	a	void	nil	B_M1_1
M1	C	a,b,c	void	nil	C_M1_1

Method invocation in the case of multiple inheritance : t.M1(a) (“t” is a variable of class C) is a method with parameter “a”. We use case statement which lists all possible cases of binding routines. In this example, the routine name “M1(a)” should be read “A_M1_1”.

```
switch (t)
{     case t = class A : A_M1_1, A_M1_2
      case t = class B : B_M1_1, A_M1_2
      case t = class C : C_M1_1, A_M1_1, A_M1_2 }
```

7 CASE STUDY

7.1 Step 1 - Mapping the Signature

We treat the methods in the OODB as one kind of attributes during the schema translation. This attribute consists of several components : name, parameter, data type, and body. Attribute name is not unique. However, a combination of an attribute name and attribute parameter is a unique key within a class object. This information will be stored in the "method" class in the frame model as a meta-data. Before storing this information, we must consider data types of return values and parameters. The data types are different between object-oriented and relational models. The object data types are mapped to the relational data types as follows:

Object Data Type		**Relational Data Type**
Integer		Integer
Char		Char
String		Varchar()
Decimal[,]		Decimal[,]
Real	=>	Float
Date		Date
Boolean		Boolean

<u>Object class</u>		**<u>Relational table</u>**	
<u>department</u>		<u>department</u>	
dept_name	char	department_oid	int
dept_no	int	dept_name	char
method:		dept_no	int
Boolean del (Integer)			

In the "method" class of the frame model, there are several attributes for handling the name conflict between Method signature and PSM signature. The PSM routine concept subsumes both functions and procedures and has been proposed for the SQL standard [8,12]. The signature includes name, parameter, and return type. All this information extracted from the definition of OODB will be stored as meta-data in the "method" class. Since the relational schema does not define the signature information in the schema definition, if the translator wants to get information regarding the routines, then it can retrieve necessary information from the meta-data as follow.

Output : Frame model ("method" class)

Method name	Class name	Parameters	Method type	Condition	Action
del	department	int	boolean		department_del

7.2 Step 2 – Source Code from Method to PSM Routine

Rule 1 - Path Expression Operand Translation

Object classes

staff		department	
id_no	int	dept_no	int
name	char	dept_name	char
dept	department		

Source OSQL : select staff.id_no, staff.name, staff.dept.dept_name
from staff
where staff.dept.dept_no = 1234;

Frame model ("attribute" class)

Attribute name	Class name	Method name	Attribute type	Default value	Cardinality
id_no	staff		int		s
name	staff		char		s
dept	staff		department		s
dept_no	department		int		s
dept_name	department		char		s

Relational tables

staff		department	
staff_oid	char	department_oid	char
id_no	int	dept_no	int
name	char	dept_name	char
dept_oid	char		

Path elements : staff => department => dept_no

The "dept_no" attribute can be located in the "department" table. The first two elements are tables. If we want to retrieve the appropriate results from these two tables, then we must perform a join operation. The translator must know how to interpret all the path elements because different path elements will provide different results of qualification statements. From the path elements, we know that the "staff" class has a composite object. The object is a class named “department”.

Translated target SQL :
```
select staff.id_no, staff.name, department.dept_name
from staff, department
where staff.dept_oid = department.department_oid and
department.dept_no = 1234;
```

Rule 2 - Set Operand Translation

Object class

office

office_no	int
office_name	char
devices_no	set{int}

Source OSQL :
```
select office.office_name
from office
where 234 in devices_no;
```

Frame model ("attribute" class)

Attribute name	Class name	Method name	Attribute type	Default value	Cardinality
office_no	office		int		s
office_name	office		char		s
device_no	office		new_devices_no		
old_devices_no	new_devices_no		int		m

Relational tables

office

office_oid	char
office_no	int
office_name	char

new_devices_no

office_oid	char
old_devices_no	int

Path elements : office => new_devices_no => old_devices_no

We can see the logic which consists of two first elements: office and new_devices_no. The linkage of these two tables relies on the primary key office_oid in office table and the foreign key office_oid in new_devices_no table. The relational statement will construct a join operation to complete the qualification statement. The attribute old_devices_no is located in the new_devices_no table and is a constraint attribute for locating all the related records in the relational table.

Translated target SQL :

```
select office.office_name
from office
where office.office_oid =
      new_devices_no.office_oid and
      old_devices_no = 234;
```

Rule 3 - Query Translation

Source OSQL :

```
select staff.id_no, staff.name, staff.dept.dept_name
from staff
where staff.dept.dept_no = 1234;
```

The attributes in the "select" clause must be located from the meta-data because they may be involved in several tables. For example, a multiple-value or a composite attribute can be referred by another relation. After locating the tables in the relational model, the system can define the table names in the "from" clause. As for the "where" clause, the system can search the path elements formed by qualification algorithm by producing a "where" clause run by RDBMS.

Translated target SQL :

```
select staff.id_no, staff.name, department.dept_name
from staff, department
where staff.dept_oid = department.department_oid and
      department.dept_no = 1234;
```

Rule 4 - Update Transaction Translation

Source OSQL :

```
update office
set devices_no = {444, 555}
where office_no = 1;
```

Firstly, the office_oid must be extracted from the office table where the office_no is equal to "1". Secondly, the old_device_no must be extracted from the new_devices_no table to replace the new value to the corresponding tuple. For locating the appropriate tuple in the new_devices_no table, the combination of values of the office_oid and old_device_no will be placed in the "where" clause.

Translated target SQL :

```
update new_devices_no
set old_devices_no = 444
where office_oid = O1 and old_device_no = 888;
update new_devices_no
set old_devices_no = 555
where office_oid = O1 and old_device_no = 999;
```

Rule 5 - Host Language Data Type Translation (for Non-volatile Data)

A mapping table of data types is created depending on specific DBMS. The mapping table in the step 1 is an example. In this rule, we shall translate host variables that have been created in the object-oriented methods for non-volatile data.

Object class			**Relational table**	
staff			staff	
id_no	int	=>	staff_oid	char
name	string		id_no	int
			name	varchar(40)

From object view :

```
void readdata ()
{     string m_name  //host variable//
      m_name = select name
      from staff
      where id_no = 1234}
```

From relational view :

```
void readdata ()
{     varchar (40) m_name
      select name
      from staff
      where id_no=1234 into
            m_name}
```

Rule 6 – Objects Inside Object Translation

Source OSQL where clause: where staff.id_no.count() >= 1;

In our example, count() is a function and is not bound by a single user-defined type. In the relational model, we can do the function before executing the where-clause. The result of this function will be stored in a temporary variable.

Translated target SQL :

```
select count (id_no) from staff into cursor temp
        :
where temp.cnt_id_no >= 1;
```

7.3 Step 3 - Invocation

Source OSQL : call del (1234) from department

The routine invocation is relied on the frame model. The "method" class of the frame model contains class name, method name, parameter list, and action name for each method. Since the relational model does not hold the object feature of polymorphism, the name of routine within a routine library must be unique. We change method names to action names which are unique in the routine library. Eventually, a specified name can be located in the routine library by the action call.

Translated target SQL : do department_del with 1234

8 PROTOTYPE

8.1 Outline of the Methodology

A prototype is provided for testing our methodology in object-oriented (Jasmine) and relational (FoxPro) systems. We loaded object-oriented schema with methods and relational schema with routines in two systems. After finishing the schema and method translations, we then ran the methods and the routines in two systems. As a result, two systems provided the same output. The data flows (see Figures 4) are as follows:

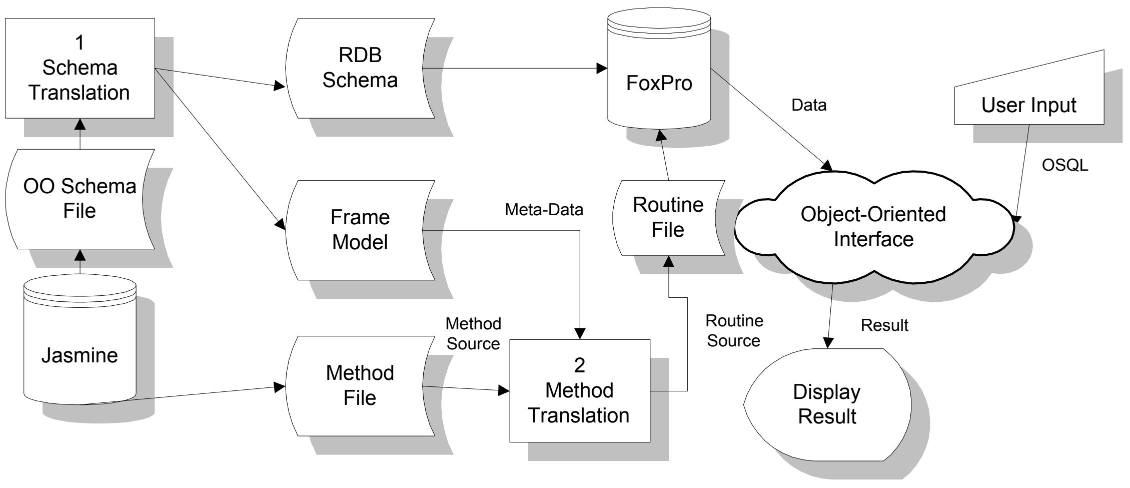

Figure 4 – Data Flow of the Prototype

The OODBMS (Jasmine) contains a schema file and a method file. In the first step, a frame model and RDB schema of the RDBMS (FoxPro) are created through the process of the schema translation. Subsequently, the method file is read and converted to a routine file by the process of the source code translation. The source code of routines will then be kept in the routine file. The final step is the invocation process.

8.2 Example Source for Method Translation

Example 1

The example shows that the source contains a simple SQL statement. We want to retrieve all the attributes in the “staff” object with “post” value of a variable “b”. The object path of the “post” attribute is simple. We can apply the Rule 3 for it.

Source Jasmine Method

```
Void findsurname (string b)
{       $string s <sval,sstat>;
        $staff p;
        $p = staff from staff where staff.post == b;
        $s = p.name;
        if (sstat == ODB_STATNIL)
        {       printf ("Surname is NIL"); }
        else
        {       printf ("Surname is %s", sval); } }
```

Translated Target FoxPro Routine

```
procedure staff_findsurname
parameters b
use staff
s = space(30)
sval = space(30)
sstat = space(1)
select * from staff where (staff.post = b) into table p
s = p.name
IF s <> null THEN
        sval = s            && non-volatile data
        sstat = “Y”
ENDIF
print_surname (sval, sstat)
close all
endproc

void print_surname (string sval, string sstat)  && C Language
{       if (sstat == ODB_STATNIL)                       && Host Language
        {       printf ("Surname is NIL"); }
        else
        {       printf ("Surname is %s", sval); }}
```

Example 2

This example shows that in the “staff” table, the “staff.id_no” and “staff.name” are attributes copying from the superclass (i.e. “person” object). The “staff.dept.dept_name” is a composite attribute connecting to the “department” object. In the where-clause, the “staff.dept.dept_no” is a composite attribute as well. We can apply Rules 1 and 3 for it.

Source Jasmine Method

```
void findsurname (integer a)
{      $string s <sval,sstat>;
       $staff p;
       $p = staff from staff where staff.dept.dept_no == a;
       $s = p.name;
       if (sstat == ODB_STATNIL)
        {      printf ("Surname is NIL"); }
       else
        {      printf ("Surname is %s", sval); }}
```

Translated Target FoxPro Routine

```
procedure staff_findsurname1
parameters a
use staff
use department
s = space(30)
sval = space(30)
sstat = space(1)
select * from staff, department;
where (staff.dept_oid = department.department_oid and department.dept_no = a);
into table p
s = p.name
IF s <> null THEN
   sval = s
   sstat = "Y"
ENDIF
close all
endproc
```

8.3 Invocation Translation

Source Invocation

```
$staff p                && Table "staff" is subclass of table "person
$p.findage (AP123)

switch (p)
{     case p = class person     : person_findage
      case p = class staff      : staff_findsurname, person_findage }
```

Meta-Data (Target Signature in the Method Class of the Frame Model)

Method_name	Class	Parameters	Method type	Action
findage	person	varchar	integer	person_findage
findsurname	staff	varchar	void	staff_findsurname

Translated Target Invocation

```
do person_findage with AP234
```

9 CONCLUSION AND FUTURE WORK

9.1 Summary

In this paper, the scope is not only on the object definition, but also on the object behaviors. The former can be defined in the OODB schema. The latter can be implemented in the object methods.

Frame model methodology employs a set of predefined classes for referring the translation to represent the object, object inheritance, and object relationship. All object definitions are stored in four system classes: Header Class, Attributes Class, Constraint Class and Method Class. Activities that relate to object behavior will query from the Method Class of the frame model. Such activities will also be remodeled and executed in RDB as defined by frame model classes.

The method translation consists of three steps. The first step is the method signature. It must be translated to routine signature and stored in a meta-data for further reference. The second step is source code translation which includes path expression operand, set operand, query statement, update transaction, host variables, and objects inside object. We use six rules assisted by the frame model for translating method source to routine source. Finally, we apply a case statement to handle invocation in the relational model.

9.2 Conclusion and Future Work

Testing on the object-oriented and relational systems has also proved the possibility of the method translation using the frame model approach. The process flows of the method translation have been shown in the section of PROTOTYPE. After translating the object methods from OODBMS into a RDBMS, the results of RDBMS are the same as the results of OODBMS. Therefore, we can conclude that our methodology of using frame model approach is feasible. In this paper, we can also bring components of an ORDBMS to reader so as to broaden your visions on database re-engineering. The contribution of the research is to provide a forward step for companies or organizations to migrate their legacy systems to object-relational systems because the significance of the finding in this research is for database interoperability and for developing an object-relational database management system. After studying the related works from other researchers, I find that in their papers, they did not cover the dynamic schema translation, especially in the method definition and source code. The OO methods play an important role on the OODBMS. Therefore, in our research, we focus not only on the translation of the object-oriented schema but also on the treatment of the object-oriented methods. It also defines new research opportunities that arise as a result of the use of method translation.

Object-Relational Database on a Relational-Database Management System can be developed according to our methodology. We want to keep our future research on creating an ORDBMS as we solve the conflicts between the Object-Oriented Database and Relational Database in this research. During my research on this topic, we found that other developers had introduced the same idea on Object-Relational Database. They have applied their methodologies implementing the OO interfaces such as DataBlade from Informix, Bolero from Software AG, and Data Cartridge from Oracle. We use the frame model to implement the OO interface because we do not have to change the existing system (i.e. Relational Model) and can easy modify the model for different situations. It can be formed as a meta-data during the schema translation and can be constructed as an application program interface for database interoperability. Also, it allows designers to combine the rules together into a class and to associate the classes in frame model as a knowledge-based system. We add the OO interface on top of the existing system. Users can apply the OO features easily in the relational system without interrupting their daily operations.

REFERENCES

1. C. Yu, Y. Zhang, W. Meng, W. Kim, G. Wang, T. Pham and S. Dao, "Translation of Object-Oriented Queries to Relational Queries", IEEE Proc. of IEEE on Data Engineering, 1995, pp. 90-97.
2. W. Meng, C. Yu, W. Kim, G. Wang, T. Pham and S. Dao, "Construction of a Relational Front-end for Object-Oriented Database Systems", Proc. of IEEE Data Engineering, 1993, pp. 476-483.
3. R.G.G. Cattell, "The Object Database Standard : ODMG-93", Morgan Kaufmann Publishers, 1996, Book, ISBN 1558603964.
4. Computer Associates Int., Inc./Fujitsu Ltd., "Jasmine User's Manual", Release 1.1, 1997, Book.
5. J. Fong and S. Huang, "Information Systems Reengineering", Springer Verlag, Chapter 7. 1997, ISBN 981-3083-15-8.
6. M. Blaha, W. Premerlani, and H. Shen, "Converting OO Models into RDBMS Schema", IEEE Software, May 1994, pp. 28-39.
7. J. Lim and D. Shin, "A Methodology of Constructing Canonical Form Database Schemas in a Multiple Heterogeneous Database Environment", Journal of Database Management, Vol.9 No.4, Fall 1998, pp. 4-11.
8. J.L. Hawkins, "FoxPro 2.5 Programmer's Reference", QUE, 1996, Book, ISBN 7302016976.
9. H. Reichgelt, "Knowledge Representation", Ablex Publishing Corporation, 1991, pp. 143-176.
10. J. Fong & P. Chitson, "Query Translation from SQL to OQL for Database Reengineering", International Journal of Information Technology, Vol. 3, No.1(1997), pp. 83-101.
11. J. Melton, "An SQL3 Snapshot", Proc. of IEEE Data Engineering, 1996, pp. 666-672.
12. C.J. Date & H. Darwen, "A Guide to the SQL Standard", Addison-Wesley, 4th edition, pp. 453-493.
13. R. Fikes and T. Kehler, "The Role of Frame-based Representation in Reasoning", Communications of the ACM, Volume 28, No.9, 1985,pp. 904-920.
14. S. DeFazio and J. Srinivasan, "Database Extensions for Complex Domains", Proc. of IEEE Data Engineering, 1996, pp. 200-202.
15. J.A. Orenstein and D.N. Kamber, "Accessing a Relational Database through an Object-Oriented Database Interface, Proc. of VLDB 95, pp. 702-705.
16. J.G. Hughes, “Object-Oriented Databases”, Prentice Hall Intl., 1991, Book, ISBN 0136298826.
17. UniSQL, Inc., “UniSQL/X User’s Manual”, Release 3.5, 1996, Book.

18. M. Blaha, W. Premerlani and H. Shen, "Converting OO Models into RDBMS Schema", IEEE Software, May 1994, pp28-39.
19. J. Fong, "Adding a Relational Interface to a Nonrelational Database", IEEE Software, September 1996, pp 89-96.
20. P. Seshadri, "Enhanced abstract data types in object-relational databases", The VLDB Journal, Volume 7, Number 3, Augeust 1998, pp 130-140.
21. A. Eisenberg and J. Melton, "SQL : 1999, formerly known as SQL3", SIGMOD Record, Volume 28, Number 1, March 1999, pp 131-138.
22. R.G.G. Cattell, "The Object Database Standard : ODMG-2.0", Morgan Kaufmann Publishers, 1997, Book, ISBN 1558604634.
23. V. Collins and D. Caviness, "A Survey of Current Object-Oriented Database", DATA BASE Advances, Volume 26, Number 1, February 1995, pp 14-29.
24. A. Snyder, "The Essence of Objects : Concepts and Terms", IEEE Software, Jan 1993, pp 31-42.
25. O. Diaz and N.W. Paton, "Extending ODBMSs Using Metaclasses", IEEE Software, May 1993, pp 40-47.
26. Peter P. Chan, "Internet : The New Frontier of Database Applications", Proceedings of the 9th International Database Conference, pp 3-9, 1999.
27. S. Huang, M. Chen, and J. Fong, "A Database Schema Integration Methodology for a Data Warehouse", Proceedings of the 9th International Database Conference, 1999, pp 415-418, ISBN 962-937-046-8.
28. S. Cheung, "Adding an Object-Oriented Interface to Relational Database Using Frame Model", Proceedings of the 9th International Database Conference, pp 138-154, 1999, ISBN 962-937-046-8.
29. K. Bennett, "Legacy System : Coping with Success", IEEE Software, January 1995, pp 19-23.
30. P. Seshadri, "Enhanced Abstract Data Types in Object-Relational Database", The VLDB Journal, Volume 7, Number 3, August 1998, pp 130-140
31. J. Fong and S. Cheung, "An Architectural Framework for Translating OODB Method to RDB Routine for ORDBMS", 2nd ACM HKPRC, 1999, p7.
32. V. Krishnamurthy, S. Banerjee, and A. Nori, "Bringing Object-Relational Technology to the Mainstream", Proceedings of the 1999 ACM SIGMOD, pp513-514, 1999.
33. M. Stonebraker, P. Brown, and D. Moore, "Object-Relational DBMSs Tracking the Next Great Wave", Second Edition, Morgan Kaufmann Publishers Inc., 1999, Book, ISBN 1558604529.
34. M. J. Carey, N. M. Mattos, and A. K. Nori, "Object-Relational Database Systems: Principles, Products and Challenges", Proceedings of the 1997 ACM SIGMOD, p502, 1997.