RETOO: Translating Relational to Object-Oriented Database Conceptual Schema

Soon Lay Ki[a], Hamidah Ibrahim[b], and Ali Mamat[c]

[a]*Faculty of Information Technology,*
Multimedia University,
Cyberjaya 63100, Selangor, Malaysia.
E-mail: lksoon@mmu.edu.my

[b,c]*Faculty of Computer Science and Information Technology,*
Universiti Putra Malaysia,
Serdang 43400, Selangor, Malaysia.
[b]*hamidah@fsktm.upm.edu.my,* [c]*ali@fsktm.upm.edu.my*

ABSTRACT

The existence of multiple, heterogeneous and autonomous databases within an organization means the globally important information exists in separate local database management systems (DBMSs), thus making the existing data inaccessible to remote users. One solution is to integrate these databases in order to form a single cohesive definition of a multi-database. Most of the integration is done by translating one database conceptual schema into another. In this paper, we will discuss a set of translation rules proposed to translate relational database conceptual schema into object-oriented database conceptual schema. A prototype called RElational-To-Object-Oriented (RETOO) has been developed based on our translation rules.

Keywords

Conceptual Modelling, Relational Database, Object-Oriented Database, Database Integration.

1.0 INTRODUCTION

The proliferation of information and communication technology has enabled organizations to own comprehensive information systems to manage their operations since decades ago. However, the computing environment in most of these contemporary organizations contains distributed, heterogeneous, and autonomous hardware and software systems, particularly database systems. The distributed and heterogeneous database systems within an organization have to be integrated in order to provide a single cohesive view of a multi-database. The translation from one database conceptual schema into another is inarguably essential in database integration.

Relational database management systems (RDBMSs) play a predominant role in today's market (Rob and Coronel, 2004; Kemper and Moerkotte, 1994). It is estimated that 80% of currently sold database management systems (DBMSs) are based on the relational model. Nevertheless, the applications where the scheme of the data is likely to change, and where the data are complex or *n*-dimensional have triggered the emerging development of object-oriented database management systems (OODBMS). OODBMSs are not only managed to provide the strengths of conventional databases, but a lot more features demanded by complex applications (Elmasri and Navathe, 2004; Rob and Coronel, 2004; Rao, 1994).

Hence, in this research project, we propose a set of translation rules to translate relational database conceptual schema into object-oriented (OO) database conceptual schema. The translation rules are also applied in a prototype called *RElational-To-Object-Oriented* (RETOO) using Java applet. More detailed information about our work can also be referred in (Soon, Ibrahim, & Mamat, 2005a; Ibrahim, Soon, & Mamat, 2005b; Soon, Ibrahim ,Mamat, & Phua, 2001; Soon, Ibrahim, Mamat, & Phua, 2000).

This paper is organised as in Section 2, previous related works are briefly discussed. While the concepts used in our translation rules as well as the details of the rules will be presented in Section 3 and 4. In Section 5, we shall discuss the result of the implementation in RETOO.

2.0 RELATED WORKS

A few works have been done on translating relational schema into OO schema. In (Siew & Wang, 2003), an approach was proposed to transform the spatial data from relational database to object-oriented database. McBrien and Poulovassilis developed a general

framework to support the schema transformation process (McBrien & Poulovassilis, 1998). Their model is demonstrated using an Entity-Relationship (ER) common data model and schema transformations on it. However, users have to understand the graph-based common data model in order to do the transformation based on the primitive transformation proposed.
Huang proposed a schema translation system, which can recapture the missing or hidden semantics of a database (Huang, Chen, Li, & Fong, 1997). The kernel of their system is an Extended Entity Relationship (EER) Data Dictionary System (DDS), which can store all the semantics of the new database system. All original database models which to be translated, must first be translated into EER model. Later, the intended model is mapped from the EER model. The reengineering process undoubtedly causes duplicate works.

In (Castellanos, Saltor, & García-Solaco, 1994; Castellanos & Saltor, 1991), a methodology to translate the relational model into Barcelona Object-Oriented Model (BLOOM model) was proposed. However, their approach tends to create extra classes, which are sometimes not necessary. Stanisic focused his work not only on schema translation, but query translation as well (Stanisic, 1999). The transformed OO database schema by Stanisic is not semantically-rich enough since it only supports two object-oriented concepts, which are inheritance and aggregation. Besides, the relationships among classes are only shown by one of the classes. In (Fong, 1997), schema translation from relational schema into object-oriented schema does no support multiple-inheritance. In addition, the relationships among classes are not specified clearly.

3.0 CONCEPTS BEHIND TRANSLATION RULES

In our approach, the translation rules are derived based on two concepts of database conceptual modelling, which are:

i. inclusion dependency
ii. key attributes and types of attributes.

3.1 Inclusion Dependency and Translation Rules

In relational database modelling, referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. In Figure 1, table *GRADE_REPORT* indicates the performance of every student. The attribute *StudID* in table *GRADE_REPORT* is a foreign key, which refers to the *StuID* in table *STUDENT*. Hence, this attribute's value in table *GRADE_REPORT* must match the *StuID*'s value of some tuples in the *STUDENT* table.

STUDENT

Name	StudID	Class	Major

GRADE_REPORT

StudID	SectionIdentifier	Grade

Figure 1: Referential Integrity Constraint

Since referential integrity constraint relates attributes across relations, it can be specified as an inclusion dependency (Elmasri & Navathe 2004). We may specify the inclusion dependency of tables in Figure 1 as follow:

ID: GRADE_REPORT.StudID $\subseteq$ *STUDENT.StudID*

The inclusion dependency above represents referential integrity constraint. Since referential integrity constraint represents the relationships among relations, or classes in object-oriented modelling, inclusion dependencies are able to represent higher-level class/subclass relationships (Elmasri & Navathe, 2004). Hence, inclusion dependency plays a major role in this research to determine the class and subclass relationships between classes.

Besides inclusion dependency, foreign keys of each relations are also needed to be further determined whether they are key attributes in that relation itself or not. Different circumstances will produce different translation results. In addition, our translation rules will take note of the types of attribute, particularly the composite attribute in the process of forming classes. The roles of all these characteristics in our translation rules are clearly demonstrated in ten translation rules, as showcased in Section 4.

4.0 RELATIONAL TO OBJECT-ORIENTED DATABASE SCHEMA TRANSLATION APPROACH

There are two main phases in our translation approach, namely the identification of classes and operations.

4.1 Classes Identification

To identify objects or class, there are four main steps in this phase:

Step 1: Translating Every Relation into a Class
The first rule stated that:

Rule 1:
If
R is a relation with attributes $A_1, A_2, \ldots A_n$,
then
create a class R with attributes $A_1, A_2, \ldots A_n$.

All the relations will be translated as classes, further determination rules will be applied in later steps. Each of these classes will only have their attributes and types of attributes being specified. Below is an example:

<u>*Relational Database Conceptual Schema:*</u>
Surgeon(*<u>SName</u>* *: String,*
Street : String,
City : String,
Country : String,
Phone-No : String)

The result of *Rule 1* in Step 1:

<u>*Object-Oriented Database Conceptual Schema:*</u>
class Surgeon
Attributes
SName, Street, City, Country: String;
Phone-No : String;
end Surgeon.

Step 2: Identifying Composite Attributes
In OO modelling, something should only be represented by a class if it represents a set of similar objects or concepts with meaningful properties and operations, which are required to be maintained by the system (Rob & Coronel, 2004).

Composite attributes represent a set of objects with meaningful simple attributes (Elmasri & Navathe, 2004; Rob & Coronel, 2004). One of the most common composite attributes is address, which normally consists of street, city, state and country. Undoubtedly, this composite attribute represents a set of similar objects or concepts. Hence,

Rule 2:
If
relation R consists of m composite attributes CA_i, where $1 \le i \le m$ and $CA_i = \{A_{i1}, A_{i2}, \ldots, A_{in}\}$ with no overlapping attributes between the CA_i, i.e. $\cap^{m}_{i=1} CA_i = \{ \}$,
then
- the attributes forming the composite attribute CA_i are taken out from class R, and are formed as a newly defined class, say T_i;
- in class R, attributes $A_{i1}, A_{i2}, \ldots, A_{in}$ forming the composite attribute CA_i are replaced by statement *RCA_i: set(T_i)*, where RCA_i is an attribute in class R referring to class T_i.

The following example demonstrates the execution of *Rule 2* on class *Surgeon* after class *Address* has been formed:

<u>*Object-Oriented Database Conceptual Schema:*</u>
class Address
Attributes
Street, City, Country : String;
end Address.

class Surgeon
Attributes
SName : String;
SAddress : set(Address);
Phone-No : String;
end Surgeon.

The attributes in any relations which form the composite attribute *Address* would be replaced by *set(Address)*.

In addition to the simple case of composite attributes above, there are two other cases, which are more complicated regarding composite attributes. To illustrate these two cases, assuming we have a relation with these attributes:

<u>*Relational Database Conceptual Schema:*</u>
Surgeon(*<u>ID_N</u>* *: String*
FName, MInit : String,
LName, Phone_No : String)

Case 1:
Assuming that there are two composite attributes in this relation, which are:
- Name : FName, MInit, LName
- Staff_No : FName, Phone_No

In Case 1, *FName* exists in both composite attributes *Name* and *Staff_No*. Attribute *FName* in *Staff_No* is actually referring to the same attribute in *Name*. The relationship between these two composite attributes will be specified clearly by our third translation rules. The attributes that form these composite attributes will be taken out from the original relation and formed as classes, same as the simple case discussed in *Rule 2*. After the translation process, we will get the following three classes:

<u>*Object-Oriented Database Conceptual Schema:*</u>
class Name
Attributes
FName, MInit, LName : String;
end Name.

class Staff_No
Attributes
FName : Name.FName;
Phone_No : String;
end Staff_No.

class Surgeon
Attributes
ID_No : String;
SName : set(Name);
Staff_No : set(Staff_No);
end Surgeon.

Case 2:
In this case, assuming there are another two sets of composite attribute in relation *Surgeon*.

- Name : FName, MInit, LName
- Staff_No : FName, LName

We will check the types of every attribute in the composite attributes. Obviously, all the attributes in both composite attributes responsible for forming the composite attributes only, thus all attributes will be taken out from the original relation. Consequently, we will get the following three classes:

<u>*Object-Oriented Database Conceptual Schema:*</u>

```
class Name
    Attributes
        FName, MInit, LName: String;
end Name.
class Staff_No
    Attributes
        FName: Name.FName;
        LName: Name.LName;
end Staff_No.

class Surgeon
    Attributes
        ID_No      : String;
        SName      : set(Name);
        Staff_No   : set(Staff_No );
        Phone-No   : String;
end Surgeon.
```

As a result of these special circumstances of composite attributes, we have produced two more rules:

Rule 3:
If
relation R consists of a composite attribute CA_1 with attributes $\{A_{11}, A_{12}, \ldots, A_{1n}\}$ and another composite attribute CA_2 with attributes $\{A_{21}, A_{22}, \ldots, A_{2m}\}$, and there exists at least an attribute in CA_2, say A_{2i}, which exists in both CA_1 and CA_2,[1]
then

- the attributes forming CA_1 are taken out from class R and formed as a newly defined class, say T_1;
- the attributes forming CA_2 are also taken out from class R and formed as another newly defined class, say T_2;
- in class T_2, attribute A_{2i} is defined as A_{2i}*: T_1. A_{2i}*;
- in class R, attributes $A_{11}, A_{12}, \ldots, A_{1n}$ are replaced by statement *RCA_1: set(T_1)*, representing composite attribute CA_1;
- similarly, statement *RCA_2: set(T_2)* is used to replace attributes $A_{21}, A_{22}, \ldots, A_{2m}$, representing composite attribute CA_2.

Rule 4:
If
relation R has a composite attribute $CA_1 = \{A_{11}, A_{12}, \ldots, A_{1n}\}$ and another composite attribute $CA_2 = \{A_{21}, A_{22}, \ldots, A_{2m}\}$ where $CA_2 \subset CA_1$ (CA_2 is a subset of CA_1),
then

- the attributes $A_{11}, A_{12}, \ldots, A_{1n}$ forming CA_1 are taken out from class R and formed as a newly defined class, say T_1;
- the attributes $A_{21}, A_{22}, \ldots, A_{2m}$ forming CA_2 are also taken out and formed as another newly defined class, say T_2;
- in class T_2, attribute A_{2i} where $1 \le i \le m$ is defined as A_{2i}*: T_1. A_{2i}*;
- in class R, attributes $A_{11}, A_{12}, \ldots, A_{1n}$ are replaced by statement *RCA_1: set(T_1)* representing composite attribute CA_1;
- in class R, statement *RCA_2: set(T_2)* is used to represent composite attribute CA_2.

Step 3: Identifying Relations Consist of Foreign Keys only
Referring to the mapping process in relational data modelling, a relation will have only foreign key attributes when the relation is formed as a result of an interaction between binary relations in M:N relationship, or as a result of *n*-ary relationship, where $n > 2$ (Elmasri & Navathe 2004; Rob & Coronel 2004).
The foreign keys, which originated from the key attributes of the participating relations in that relationship will form the primary keys of this newly formed relation. For this kind of relations, we will treat them as an object resulted from the interaction between or among the classes that the foreign key attributes reference to. The translation is reflected in *Rule 5*:

Rule 5:
If
relation R consists of n attributes $A_1, A_2, \ldots, A_n$ where each A_i is the foreign key that reference to relation U_i, where $1 \le i \le n$,
then

- class R is treated as interactions of all the classes $\{U_1, U_2, \ldots, U_n\}$;
- in class U_i, statements {*R: set(U_1) inverse is U_1.R, R: set(U_2) inverse is U_2.R, …, R: set(U_n) inverse is U_n.R* } – {*R: set(U_i) inverse is U_i.R*} are stated;
- class R is abolished.

[1] If there is an attribute in CA_2, say A_{2j} which is a simple attribute by itself, then in class T_2, attribute A_{2j} is defined as *A_{2j}:R.A_{2j}*; and attribute A_{2j} will remain in class R.

The example below illustrates this translation step.

Relational Database Conceptual Schema:
Paper(P#, Title, Issue# : String,
Institute_Name, Vol# : String)

Author(AName, Nationality : String,
Date_of_Birth : Date)

Writes(P#, AName : String)

Inclusion Dependencies:
ID: Writes.P# Í Paper.P#
ID: Writes.AName Í Author.Aname

Writes relation contains only two foreign key attributes where *P#* refers to the *P#* in relation *Paper*, and *AName* refers to the *AName* in relation *Author*. Hence, relation *Writes* is actually representing the interaction between relations *Paper* and *Author*. After the translation of *Rule 5*, class *Writes*, which is formed after *Rule 1* in translation step 1 is abolished.

Object-Oriented Database Conceptual Schema:
class Paper
  Attributes
    P#, Title, Issue# : String;
    Institute_Name, Vol#: String;
    Written_by : set(Author)
      inverse is Author.write;
end Paper.

class Author
  Attributes
    AName, Nationality : String;
    Date_of_Birth : Date;
    Write: set(Paper) inverse is Paper.written_by;
end Author.

Step 4: Identifying Foreign Keys and The Candidate Keys Being Referenced

There are two main possibilities identified regarding the referential integrity as shown in Table 1. We categorise this step into case 1 and case 2, whereby case 1 is when both the foreign key and the key being referenced are key attributes. While case 2 represents the occurrence of foreign key as a non-key attribute. For the purpose of this project, we regard composite primary key as a key attribute that consists of more than one simple attributes.

Table 1: Foreign Key

Foreign Key	Candidate Key Being Referenced
Key Attribute	Key Attribute
Non-Key Attribute	Key Attribute

Case 1:
In this case, we can further divide it into another four categories, as shown in Table 2:

Table 2: Four Categories of Case 1

Foreign Key (Key Attribute)	Candidate Key Being Referenced
Simple Primary Key	Simple Primary Key
Composite Primary Key	Composite Primary Key
Composite Primary Key	Simple Primary Key
Part-of Composite Primary Key	Simple/Composite Primary Key

In relational database modelling, the key attribute is an attribute whose values are used to identify each individual entity uniquely (Elmasri & Navathe, 2004; Rob & Coronel, 2004). Specifying that an attribute is a key of an entity type means that the preceding uniqueness property must hold for every extension of that entity type (Elmasri &d Navathe, 2004; Rob & Coronel, 2004). This key constraint is derived from the properties of the miniworld that the database represents (Elmasri & Navathe, 2004).

The key constraint in relational modelling indicates that when the key attribute of a relation R_1 is a foreign key, this relation refers to the whole relation R_2 that contains the key being referenced. Hence, R_1 is an instance of R_2 whereby besides the attributes in R_2, R_1 has its extra attributes. In OO modelling, this situation is similar to inheritance. A subclass is inherited from a superclass if the subclass "is-an" instance of the superclass.

For category one, if both the foreign key and the candidate key being referenced are simple primary key attributes of the relations, the foreign key's relation is considered as an inheritance of the relation which is being referenced. This applies to the second category where both of the foreign key and the key being referenced are composite primary keys. As stated in *Rule 6*:

Rule 6:
If
  both the foreign key in relation R and the candidate key being referenced in relation V are simple primary key attributes or composite primary keys,
then
- class R is treated as an inheritance of class V;
- statement *inherit V* is included in class R;
- statement *inherited_by T* is included in class V.

For example, the *SName* attribute in *Consultant* is the foreign key, which refers to the primary key of

Surgeon. In this case, we can say that the *Consultant* “is-a” *Surgeon*.

Relational Database Conceptual Schema:
Surgeon(SName, Street, City : String,
Country, Phone_No : String)

Consultant(SName, Speciality : String)

Inclusion Dependency
ID: Consultant.SName Í Surgeon. SName

After translation, we shall get the following OO schema:

Object-Oriented Database Conceptual Schema:
class Surgeon
inherited_by Consultant
Attributes
SName : String;
SAddress : set(Address);
Phone_No : String;
end Surgeon.

class Consultant
inherit Surgeon
Attributes
Speciality : String;
end Consultant.

There are certain relations with more than one foreign key and all the foreign keys formed the primary key attributes of the relations. Besides these foreign keys, these relations might also have their own attribute(s). If the subclass “is-an” instance of the superclasses, the relationships among these relations are translated as multiple inheritance. Assuming in a stationery manufacturing factory, it produces a *Notebook*, which is a *CommercialProduct* and also a *Gift* for customers:

Relational Database Conceptual Schema:
CommercialProduct(CommercialID : String,
Packaging : String,
Price : Integer)

Gift(GiftID, Category : String,
Coupon: Integer)

Notebook (CommercialID, GiftID : String,
Size: Integer)

Inclusion Dependencies
ID: Notebook.CommercialID Í
CommercialProduct. CommercialID
ID: Notebook.GiftID Í Gift. GiftID

The *Notebook* “is-a” *CommercialProduct* and also “is-a” *Gift* to the manufacturer. As a result, these three classes will be translated as follow:

Object-Oriented Database Conceptual Schema:
class CommercialProduct
inherited_by Notebook
Attributes
CommercialID : String;
Price : Integer;
Packaging : String;
end CommercialProduct.

class Gift
inherited_by Notebook
Attributes
GiftID, Category : String;
Coupon : Integer;
end Gift.

class Notebook
inherit CommercialProduct
inherit Gift
Attributes
Size : Integer;
end Notebook.

As mentioned above, in OO modelling, a subclass is inherited from a superclass only if the relationship between the subclass and the superclass is “is-a” relationship. Refer to the example below:

Relational Database Conceptual Schema:
Programmer(SSN, Salary, Sex : String,
BDate : Date)

Project(P#, PName : String,
StartDate, DueDate : Date)

Works_On(SSN, P# : String,
Hours : Integer)

Inclusion Dependencies
ID: Works_On.SSN Í Programmer. SSN
ID: Works_On.P# Í Project. P#

Works_On is neither “is-a” *Programmer* nor “is-a” *Project*. In fact, *Works_On* would be more suitable to be identified as an aggregation or assembler of the two classes. According to the mapping process in relational modelling, *Works_On* shows the M:N relationship between *Programmer* and *Project*. While the attribute *Hours* is an attribute obtained from the relationship between *Programmer* and *Project*.

From the aggregation perspective in OO modelling, class *Works_On* does not concern about the representation details of *Programmer* and *Project*. All the properties of the *Programmer* and the *Project* associated with a particular *Works_On* occurrence are encapsulated by the class and may be accessed without explicit joins.

Thus, not all relations with foreign keys as composite primary key are translated as multiple inherited class. The translation result would be:

<u>*Object-Oriented Database Conceptual Schema:*</u>

class Programmer
participate_in Works_On
Attributes
SSN, Salary, Sex : String;
BDate : Date;
end Programmer.

class Project
participate_in Works_On
Attributes
P#, PName : String;
StartDate, DueDate : Date;
end Project.

class Works_On
assemble Programmer
assemble Project
Attributes
Hours :Integer;
end Works_On.

For third category, *Rule 7* and *Rule 8* are formed:

Rule 7:
If
relation R has a set of foreign keys $\{fk_1, fk_2, \ldots, fk_n\}$ where $n > 1$ and fk_i where $1 \le i \le n$ formed the primary key of R, and after being translated into class R, class R is an instance of the classes $C_1, C_2, \ldots, C_m$ where its foreign keys referenced to, i.e. $R.fk_i \subseteq C_j.pk$[2], where pk is the primary key of C_j,
then
- class R is treated as an inheritance of classes $C_1, C_2, \ldots, C_m$;
- in class R, statements *inherit* C_j, where $1 \le j \le m$ are included;
- statements *inherited_by R* is included in classes $C_1, C_2, \ldots, C_m$.

Rule 8:
If
relation R has a set of foreign keys $\{fk_1, fk_2, \ldots, fk_n\}$ where $n > 1$ and fk_i where $1 \le i \le n$ formed the primary key of R, and after being translated into class R, class R is an aggregation of classes $C_1, C_2, \ldots, C_m$ where its foreign keys referenced to, i.e. $R.fk_i \subseteq C_j.pk$, where pk is the primary key of C_j,
then
- class R is treated as an aggregation of classes C1, C2, …, Cm;
- statements assemble Cj where $1 \le j \le m$ are included in class R;
- statement participate_in R is included in classes C1, C2, …, Cm.

[2] The symbol $\subseteq$ shows the inclusion dependency.

The fourth category of this case indicates another situation where the foreign key is a part-of primary key. The candidate key(s) being referenced can be simple or composite primary key(s). In relational database modelling, this happens when the relation that contains the foreign key(s) is a weak entity. The key attribute of the parent entity is included as a foreign key in the weak entity and will be part of the key attribute in the weak entity. *Rule 9* is derived such that:

Rule 9:
If
part of the primary key of relation R is a foreign key attribute, which refers to a relation Q,
then
- class R is treated as a weak entity, which depends on class Q;
- statement *depend Q* is included in class R;
- statement *has_dependent R* is included in class Q.

An example is shown below, the class *Children* is a weak entity that depends on its parent entity *Employee*.

<u>*Relational Database Conceptual Schema:*</u>
Employee(<u>SSN#</u>, Name, Gender : String)
Children(<u>SSN#, Child_Name</u>, Gender : String, Age : Integer)

<u>*Inclusion Dependency*</u>
ID: Children.SSN# Í Employee. SSN#

The following two classes are obtained after *Rule 9*:

<u>*Object-Oriented Database Conceptual Schema:*</u>

class Employee
has_dependent Children
Attributes
SSN#, Name, Gender: String;
end Employee.

class Children
depend Employee
Attributes
Child_Name Gender: String;
Age : Integer;
end Children.

Case 2:

In relational database mapping process, for each regular binary 1:1 and 1:N relationship type *R*, we should identify the relation *S* that represents the participating entity type at the full participation or N-side of the relationship type. Then, include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*, as showcased in Case 2 of Table 1.

The mapping process mentioned above shows that the existence of the non-key attribute in relation *S* that refers to the key attribute of relation *T* means that the foreign key in *S* is merely referring to relation *T* and not an instance of relation *T* or even assembling relation *T*. The existence of this foreign key as non-key attribute is regarded as an interaction between *S* and *T*.

Below is an example demonstrating our approach:

<u>Relational Database Conceptual Schema</u>:
Employee(<u>SSN</u>, Sex, Salary, DeptNo: String,
BDate : Date)

Department(<u>DeptNo</u>, DName, Location: String)

<u>Inclusion Dependency</u>
ID: Employee.DeptNo Í Department.DeptNo

After being translated:

<u>Object-Oriented Database Conceptual Schema:</u>
class Employee
Attributes
SSN, Sex, Salary : String;
BDate : Date;
Work_in : set(Department)
inverse is Department.Worked_by;
end Employee.

class Department
Attributes
DeptNo, DName, Location: String;
Worked_by: set(Employee)
inverse is Employee.Work_in;
end Department.

Based on our finding on Case 2, the last translation rule in our approach is:

Rule 10:
If
relation R has a foreign key f which is not a key attribute, that refers to a relation P,
then
- attribute f shows the interaction between class R and class P;
- in class R, statement *f: set(P) inverse is P.R* replaces attribute f;
- in class P, statement *R: set(R) inverse is R.f* is included.

4.2 Operations Identifications

In OO modelling, there are basically three categories of operations for each class:

1. Constructor/destructor functions
2. Accessor/query functions
3. Transformer/update functions

User intervention in this phase is crucial as the information of operations for each object or relation is not provided in the relational data model. By default, we suggest two operations for every class, which are the constructor and destructor operations. Nevertheless, users will still have the flexibility whether to accept the default operations or not. For other necessary functions, users will need to provide the information to RETOO though.
Showing below is an example of these two basic operations applied to a class:

<u>Object-Oriented Database Conceptual Schema:</u>
class Lecture
Attributes
LectureID : String;
location : set(Address);
……
Methods
create(…);
destroy(…);
end Lecture.

5.0 RETOO IMPLEMENTATION

We have developed a prototype based on the translation rules using Java. The tools have been tested with numerous cases and it is evident that our translation rules work well in translating relational to OO database conceptual schema. Figure 2 shows the original relational database conceptual schema which will be translated. While Figure 3 presents a comparison between our translated OO model and BLOOM model.

employee(<u>ss#</u>, dept, salary)
department(<u>d_name</u>, location, budget)
ID:employee.dept ⊆ department.d_name

Figure 2: Relational Database Conceptual Schema

In this example, class *privileged* has been created in BLOOM model. According to Castellanos et al., this class is created because *employee.dept* is not null-constrained. Therefore, it can exist as null value. For those employees whose dept attribute is null, they are considered as "privileged-employees".

The referential integrity constraint in relational database modelling specifies that a foreign key can either exists as a value of the candidate key it reference to or exists as null. Obviously, the forming of new class *privileged* is actually not necessary since the existence of null value for dept is acceptable. In our approach, the existence of dept in class employee is represented as *work_in:department.worked_by*, demonstrating the interaction between these two classes.

```
BLOOM
class employee                    class privileged
    subclass privileged               superclass    employee
    id ss#                            exception_on dept
    atrs      department          end_class
              salary
end_class                         class department
                                      s_agg_of manager
                                      id d_name
                                      atrs  budget
                                  end_class
```

```
RETOO
class department
    Attributes
      d_name, location, budget       :     string;
      worked_by    : set(employee) inverse is
                     employee.work_in;
end department.

class employee
    Attributes
      ss#          : string;
      work_in      : set(department) inverse is
                     department.worked_by;
      salary       : integer;
  end employee.
```

Figure 3: Comparing RETOO Result and BLOOM

6.0 CONCLUSION

In this research project, we have proposed a set of translation rules to translate relational database conceptual schema to object-oriented database conceptual schema. The rules have been implemented in a prototype called RETOO by using Java applet. The relational semantics are well-preserved in the translated object-oriented conceptual schema.

Currently, RETOO needs users to enter the relational schema, including the referential integrity, key attributes and types of the attributes for the translation process. Our plan is to improve RETOO by minimizing user's work in entering the information regarding relational conceptual schema. A more autonomous translator, which is similar to a compiler that is able to read and capture the information regarding relational conceptual schema by itself shall be the future direction of this field. Besides having a fix set of translation rules, we would also like to explore the possibility of incorporating description logic in our database schema translation.

REFERENCES

Castellanos, M., Saltor, F., & García-Solaco, M. (1994). Semantically Enriching Relational Databases into an Object Oriented Semantic Model. In Proceedings Of *the 5th International Conference on Database and Expert Systems Applications*. (pp. 125-134). London, UK.

Castellanos, M. & Saltor, F. (1991). Semantic Enrichment of Database Schemas: An Object Oriented Approach. In Proceedings of *the 1st International Workshop on interoperability in Multimedia Systems*. (pp. 71-78). Kyoto, Japan.

Elmasri, R. & Navathe, S.B. (2004). Fundamentals of Database Systems. United States of America: Addison Wesley.

Fong, J. (1997). Converting Relational to Object-Oriented Databases. *The ACM SIGMOID Record*, 26(1): 53 – 58.

Huang, S.M., Chen, H.H., Li, C.H., & Fong, J. (1997). *A Data Dictionary System Approach for Database Schema Translation*. Publication of IEEE, 3966-3971.

Soon, L.K., Ibrahim, H. & Mamat, A.. (2005). Constructing Object-Oriented Classes from Relations. In Proceedings of *MMU International Symposium on Information and Communication Technologies 2005*. (pp. TS13 17-20). Petaling Jaya, Malaysia.

Ibrahim, H., Soon, L.K. & Mamat, A. (2005). Developing Translation Rules for Converting Relational to Object-Oriented Database Conceptual Schema. *Pertanika Journal of Science and Technology*, 13 (1): 1-21.

Kemper, A., & Moerkotte, G.. (1994). *Object-Oriented Database Management, Applications in Engineering and Computer Science*. United States of America: Prentice Hall Inc.

McBrien, P., & Poulovassilis, A.. (1998). Automatic Migration and Wrapping of Database Applications – A Schema Transformation Approach. *Department of Computer Science Technical Report*, King's College London.

Rao, B.R.. (1994). *Object-Oriented Databases, Technology, Applications, and Products*. United States of America: McGraw-Hill, Inc.

Rob, P. & Coronel, C. (2004). *Database Systems: Design, Implementation, & Management*. United States of America: Thomson Learning.

Siew, T.K., & Wang, Y.C. (2003). An Object-Oriented Approach for Transformation of Spatial Data from Relational Database to Object-Oriented Database. Proc. Of *The International Conference on Asian Digital Libraries* (ICADL), Kuala Lumpur, Malaysia, 533 – 543, Lecture Notes in Computer Science, Springer-Verlag.

Soon, L.K., Ibrahim, H., Mamat, A., & Phua, C.S. (2001). Translating Relational Model to Object-Oriented Model. In Proceedings Of *The International Conference on Information Technology and Multimedia*. (pp. 525-532).

Selangor, Malaysia: The International Conference on Information Technology and Multimedia.

Soon, L.K., Ibrahim, H., Mamat, A., & Phua, C.S. (2000). Relational-to-Object-Oriented Database Schema Translation. In Proceedings of *Information Technology Colloquium* (INTEC) 2000. (pp. 99-104). Selangor, Malaysia: UPM Information Technology Colloquium

Stanisic, P. (1999). Database Transformation from Relational to Object-Oriented Database and Corresponding Query Translation. In Proceedings Of *Workshop on Computer Science and Information Technology*. (pp. 199-208).