Interoperable Database Systems (DS-5) (A-25)
D.K. Hsiao, E.J. Neuhold and R. Sacks-Davis (Editors)
Elsevier Science Publishers B.V. (North-Holland)
1993 IFIP



# Translating Relational Schema With Constraints Into OODB Schema

Ling-Ling Yan[a] and Tok-Wang Ling[b]

[a]Institute of Systems Science, National University of Singapore, email: lingling@iss.nus.sg

[b]Dept. of Information Systems and Computer Science, National University of Singapore

**Abstract**

We consider the problem of translating a relational database schema into that of an object-oriented database. Our approach is designed to extract semantics by investigating into the structure of relations, their key constraints, and inclusion/referential constraints. These constraints can be existing ones from the schema or they can be given by users who know about the data characteristics in the relational database. In this way, our translator can achieve semantic enhancement by explicitly representing implicit or missing semantics in a relational database schema. The major features of our translator include identifying relation clusters representing object classes, identifying ID-dependencies(complex object classes), identifying ISA hierarchy among objects, generation of object identifiers, and identifying inter-object relationships.

Keyword Code: H.2.5
Keyword: Heterogeneous Databases

## 1 Introduction

Database schema translation has been an interesting problem. Significant research has been done to address the issue [17, 9, 7, 8, 14, 19]. Recently, with the research into heterogeneous database, the issue finds an important role in building mappings from component database schemas, in local data models, into those in canonical data model [15]. Due to its semantic richness and flexibility in modeling, object-oriented data model(OODM) is usually chosen as the canonical data model. In this paper, we deal with the specific problem of schema translation from relational schema into an Object Oriented Data Base (OODB) schema.

Semantically, relational data model(RDM) is not as expressive as any of the object-oriented data models. This causes certain semantics of application domain to be represented implicitly or missing from the relational database schema. For example, ISA relationship, IS-PART-OF relationship(complex object class) may be represented as plain relationships without special meanings. Moreover, for the purpose of conforming to First Normal Form(1NF) and avoiding large amount of data redundancy, e.g. those caused by multivalue dependency, one object class may have been represented by multiple relations. For a good OO representation of the current relational database schema to make the later semantic based multidatabase view construction easier, the translation should make explicit the semantics that was implicitly represented in relational schema. Also, a translator should give DBA a chance to specify missing semantics that could not be represented in a relational schema but is now possible to be represented by an OODB schema. This does not mean new data, but the description of a more meaningful way of interpreting the existing data in the relational databases.

In this paper, we present a translator that satisfies the above requirements. The translator will take a relational database schema, allow users to specify more semantics based on the data in the databases, and produce an OODB schema. The extra semantics will be accepted in two forms: primary/candidate key constraints and inclusion constraints. Notice that all these constraints should be derived from the data characteristics in the relational database. The main features of this translator consist of the followings:

1. Identify clusters of relations that represent object classes. A C++-like class definition will also be generated.
2. Identify identifier dependencies. Identifier dependency represents complex object class.
3. Identify ISA relationships among object classes.
4. Generate object identifiers(oids) for all the identified object classes.
5. Identify relationships among objects.

The translator starts from a relational schema with key constraints and a set of inclusion constraints and produce an OODB schema. In this paper, we present the working of our translation algorithm by giving examples to show how it achieves the five objectives mentioned above.

# 2 Background

Schema translation in heterogeneous database context is an important problem to solve. With the choice of OODM as the canonical data model, any local database schema that is not in OODM must be translated into an OODB schema so that a multidatabase

view can be constructed for users to access data from heterogeneous databases [15]. We concentrate on the specific problem of translating relational database schema into an OODB schema. The most simple approach to this problem is to translate each relation into an object class in the target schema. The problem with this approach is that it is semantically weak. For example, the semantics of referential constraint, which is supported by most modern relational databases such as DB2 [18], cannot be represented. This means that the OO translation is semantically weaker than the original relational database schema. Obviously, this will be a bad translation.

Schema translation is not a new issue. Relationship among different data models has been investigated on a formal basis by [17] and [9]. In [17], a detailed discussion on the various issues involved in data model mapping is given. The problem of mapping between relational and network data models is solved to detail. The authors also comment that very little formalism exists for general data model mapping. Most of the time, we only see ad hoc mapping specifications between specific pairs of data models. The formal approach of [9] is based on a denotational semantics of data model equivalence. Every data model is mapped into a denotational representation based on which equivalence and mapping can be generally defined and strictly specified among different data models.

Both [17] and [9] aim to solve the problem by defining a mapping based on data model constructs. While the idea may lead to an ultimate solution to data model mapping problems, it is not clear how to apply this idea to construct mappings between data models that are different in their expressive power of semantics. In the context of relational to OODB schema translation, we realize that OODM has a stronger power in representing real world semantics. This can be understood from two observations. First, an OODB schema as the result of a translation from a relational schema may contain more semantics. For example, in an Employee database, the fact that every Manager is also an Employee might be maintained by all the application programs but can now be represented by a subclass relationship in the OODB schema. Second, each construct in relational data model may have several possible interpretations in OODM. To remove this ambiguity, constraints must be taken into consideration. We believe that research to further identify the data model mapping issues on a semantic(rather than syntactic) level is necessary.

The work presented in this paper is not yet another general data model mapping method, rather, it is a specific method designed for the context of translating relational schema into OODB schema. Especially, we give rigorous rules which use key constraints and inclusion constraints to identify semantics from the relational database. The nature of our work is similar to that of [7] and [5]. However, the work in [7] is not detailed enough. For example, entity fragmentation caused by multivalued dependency is not considered, nor is ISA relationship considered. In [5], the semantics of various types of inclusion dependency is analyzed. Compared with our approach, [5] has tendency of generating large number of "missing" classes. Moreover, the generated object class definition is rather complicated. These may give difficulty to later query translation and data integration, which are necessary steps in the context of heterogeneous database system

development.

The major objective of our approach is to identify as much as possible OO semantics from the relational database. This implies two sub-goals. First, the semantics that was represented implicitly or ambiguously in the relational database schema should be made explicit as long as the OODM is powerful enough to represent it. For example, the ISA relationship in relational database. Second, specification of missing semantics should be accepted to make the OO translation correct. We accept extra semantics in the form of primary/candidate key constraints and inclusion/referential constraints. These constraints are basically known data characteristics that are not declared in the relational schema for one reason or another. The specification of them will direct the translation to the right track without affecting the existing data repository in the underlying database.

Generally, a relational DBMS supports the specification of primary keys but not candidate keys. Candidate key constraints, if any, should be provided to the translator as extra semantics. Some relational DBMSs support declarative specification of referential constraints, e.g. in DB2 [18]. The concept of inclusion constraint, of which referential constraint is a special case, is not generally supported. An **inclusion constraint** is denoted in the following form:

$$R_1[P_1] \subseteq R_2[P_2]$$

where $R_1$, $R_2$ are relations,$P_1$ and $P_2$ are (sets of) attributes in relations $R_1$ and $R_2$, respectively. This constraint states that for $r_1$ and $r_2$, instances of relation schemas $R_1$ and $R_2$, respectively, the following always holds: $r_1[P_1] \subseteq r_2[P_2]$. Consider two relations R and R′. If there exists a primary or candidate key of R, P, and a set of attributes of R′, A′, s.t. R′[A′] ⊆ R[P], we say **R′ references R**. Moreover, A′ in R′ is a **foreign key**.

The concept of inclusion constraint has been widely investigated [4, 11, 12, 13]. However, the impact of this concept on semantic modeling for relational database is not very clear.

The OODM we use will contain generally accepted constructs for object-oriented data models. Currently, we employ a C++-like syntax for presenting the result of translation.

# 3 Object class as a cluster of relations

An object class in the result OODB schema corresponds to a named cluster of relations from the underlying relational database. Each instance of the class has a unique identifier. In this section, we present the way in which the algorithm identifies the clusters and their names. Generation of object identifiers will be discussed in section 6.

First, we define the concept of *main class relation* and its *component relations*. Intuitively, a main class relation represents the core part of an object class while a component

relation represents other properties of an object class.

### Main Class Relation

Consider a relation R. R is a main class relation if one of the following cases is true:

**case1:** R is not involved in any inclusion constraints.

This is the case where a relation is stand-alone. The semantics of such relation is taken as an object class that is not involved in any relationship. In fact, such relations may represent certain functional and/or multivalue dependencies. These semantics is implicit and it cannot be made explicit in the target schema.

**case2** : The followings hold:

1. There exists a relation R′ that references R.
2. The primary key of R does not contain more than one disjoint foreign key.
3. There exists no inclusion dependency whose right hand side is a proper subset of the primary key of relation R.

Condition 1 says that relation R is referenced by a relation. Condition 2 says that the primary key of R cannot be decomposed into more than one foreign keys. Condition 3 will become clear in section 4.

**case3:** R is identified as a main relation by ID-dependency identification rule as discussed in section 4.

### Component Relation

Let R be a main class relation. Relation $R_1$ is a component relation of R if the followings hold:

1. No relation references $R_1$.
2. The primary key of $R_1$ does not contain more than one disjoint foreign key.
3. Let the primary key of $R_1$ be $K_1$. Then there exist $P_1$, $P_1 \subseteq K_1$ or $P_1$ is a candidate key of $R_1$, such that $R_1[P_1] \subseteq R[P]$, where P is a primary or candidate key of relation R.

Each main class relation and all its component relations form a relation cluster that represents an object class. The name of the object class will be the name of the main class relation. This is shown by the following example.

**Example 3.1.** Consider the following relations with keys underlined:

$Person(\underline{Pno}, name, age)$

$PersonPhone(\underline{Pno, phoneNo})$

$DrivingLicense(\underline{Pno}, licenseNo)$

$Parent(\underline{Pno, childPno})$

The primary key of relation Person is Pno. Relation PersonPhone is all-key, meaning that one person can have several phone numbers. The primary key of relation DrivingLicense is Pno, meaning that one person can have only one driving license. Relation Parent is all-key. Also assume that we have the following inclusion constraints:

$PersonPhone[Pno] \subseteq Person[Pno]$

$DrivingLicense[Pno] \subseteq Person[Pno]$

$Parent[Pno] \subseteq Person[Pno]$

$Parent[childPno] \subseteq Person[Pno]$

Using our approach, Person will be identified as a main class relation. The relations PersonPhone and DrivingLicense are not main class relations because no relation references them but they themselves reference relation Person. By definition, they are the component relation of the main class relation Person. Relation Parent is neither a main class relation nor a component relation because its primary key can be decomposed into two disjoint foreign keys, namely, Pno and childPno. As it turns out later, relation Parent will be considered as a relationship object. For now we identify the following relation cluster:

Person = {Person, PersonPhone, DrivingLicense}

This relation cluster will give rise to an object class Person with the following definition:

```
class Person {
        string Pno;
        string name;
        integer age;
        integer DrivingLicense_licenseNo;
        setof(string) PersonPhone_phoneNo;
};
```

A detailed description of the generation of this definition will not be given here. Instead, we give some intuitive observations on how the algorithm works. Notice that sometimes, attribute name will be prefixed by the name of the relation it comes from, e.g. the attribute DrivlingLicense_licenseNo and PersonPhone_phoneNo. By doing so, we can make sure that all attribute names are unique and hence avoid possible confusion in attribute naming. Another important thing to notice is that set-valued attributes will be identified. In this example, it is obvious that the attribute PersonPhone_phoneNo should be set-valued since the relation PersonPhone is all-key. This is reflected in the class definition generated.

A component relation can never be a main class relation. A main class relation is either "stand alone", not involved in any inclusion dependencies, or is referenced by some other relations, while a component relation always references another relation but is not referenced by any other relation. Intuitively, this reflect a criteria we employ for identifying the "cores" of objects: a relation may give rise to an object class in the result OODB schema only if it is stand alone or referenced by others. This criteria will help to avoid proliferation of object classes and encourage possible merging of relations. As shown later, this will also help in avoiding unnecessary ISA links between classes. In the above example, if we did not employ this criteria, relation Driving_license may give rise to an independent class which is not necessary in this case. We'll further discuss the effect of this criteria on ISA hierarchy identification in section 5.

A main class R will ultimately give rise to an object class with the same name. We refer to this object class as object R hereafter.

# 4 Identifier Dependency and Complex Object

Identifier dependency(ID-dependency) is a term from entity relationship approach [6]. An entity B is identifier dependent on entity A if it does not have its own key so that it has to depend on the the identification of A in order to be identified. This is best shown by the example of Wards in Hospital. Usually, a Ward is identified by the Hospital it belongs to and its room number. The identification of Ward depends on that of Hospital. This dependency also implies some existence dependency i.e. a Ward cannot exist without a Hospital.

We consider the treatment of ID-dependency because it happens quite often in relational database. This is partially due to the fact that a relational database design is usually done by using ER based technologies. Moreover, the nature of the relational database model imposes value-based object identification. If an object does not really have an identification of its own, it has to be identified based on ID-dependency. However, in OO terms, this is the typical case of complex object, for example, object Ward should be a component object of the object Hospital. In OODM, this IS-PART-OF relationship can be based on oid. Hence Ward can appear as an independent object, which

can then participate in relationships with other objects, while its existence depends on the existence of another object, Hospital. In [3], such dependency is indicated by a keyword "own". We'll borrow this notion to emphasize our points.

In [7], similar cases are considered. [7] also identifies the weakness(or dependency) implied by such cases but their treatment is not correct when applied to Ward-Hospital example. The approach in [7] does not have enough justification except ad hoc examples. It is not clear whether or not "entity discriminator" can be used as identifier attributes. Compared with [7] approach, our approach does not make any assumptions on the "dangling keys", rather, we treat them as ordinary properties of the dependent entity.

In our approach, we identify ID-dependency and represent it as a special inter-object relationship. The ID-dependency is identified by the rule below:

**ID-Dependency Identification Rule**

Let $R_0$ be a main class relation with primary key $K_0$. Consider a relation R, with primary key K, that satisfies the followings:

1. There exist $K' \subset K$, s.t. $R[K'] \subseteq R_0[K_0]$.
2. The primary key of R does not contain more than one disjoint foreign key.
3. There exists a relation that references R.

Then R is identified as a main class relation. Moreover, object class R is ID-dependent on object class $R_0$ via the inclusion dependency $R[K'] \subseteq R_0[K_0]$.

Notice that without condition 3 in the above rule, relation R will be taken as a component relation of relation $R_0$. Condition 3 basically says that R qualifies to be an independent object class.

**Example 4.1.** Hospital-Ward example. We continue from Example 3.1, consider the following relations with keys underlined:

$Hospital(\underline{Hname}, address)$

$Ward(\underline{Hname, wardNo}, doctor)$

$WardPatient(\underline{Hname, wardNo, PatientPno})$

Assume we have the following inclusion constraints:

$Ward[Hname] \subseteq Hospital[Hname]$

$WardPatient[Hname, wardNo] \subseteq Ward[Hname, wardNo]$

$WardPatient[PatientPno] \subseteq Person[Pno]$

Applying the rules given above, relation Ward and relation Hospital will be identified as main class relations. Moreover, object class Ward is ID dependent on object class Hospital via Ward[Hname] $\subseteq$ Hospital[Hname]. The translation will produce the following two class definitions:

```
class Hospital {
      string Hname;
      string address;
      own setof(Ward) Ward;
      };

class Ward {
      string Hname;
      string wardNo;
      string doctor;
      };
```

Notice that the attribute Ward in class Hospital. The keyword "own" preceding the specification indicates the existence dependency of Ward on Hospital. Also notice that the type of this attribute is "set of objects". A detailed procedure is implemented by the complete algorithm to generate these definition. This will not be further discussed in this paper.

# 5 ISA Hierarchy

After identifying all the main class relations and the relation clusters induced by them, we can now identify the ISA relationship among the classes by using the following rule.

### ISA Hierarchy Identification Rule

Consider two main class relations $R_1$ and $R_2$. If there exist $P_1$ and $P_2$, keys in relations $R_1$ and $R_2$, respectively, such that $R_1[P_1] \subseteq R_2[P_2]$, then the following is true for object classes $R_1$ and $R_2$:

$R_1$ *ISA* $R_2$ *via* $R_1[P_1] \subseteq R_2[P_2]$.

Intuitively, if all the instances of class $R_1$ participate in class $R_2$, we take it as an ISA relationship between the two classes. We note two points in the above mentioned rule. First, $R_1$ and $R_2$ must be both main class relations. Second, the inclusion constraint must be between the keys of the relations. A relation can not be a component relation and a main class relation at the same time. This together with the fact that ISA

link can only be identified between main class relations makes sure that no ambiguity exist as for whether two relations are related by sub-class link or component relation link.

We illustrate the usage of the rule by the following example.

**Example 5.1.** Consider the following relations with keys underlined:

$Person(\underline{Pno}, name, age)$;

$DrivingLicense(\underline{Pno}, licenseNo)$;

$PersonPhone(\underline{Pno, phoneNo})$;

$Employee(\underline{Eno}, \underline{Pno}, dateOfJoin)$;

$ProjStaff(\underline{ProjNo, Eno}, position)$;

$SalaryHistory(\underline{Eno, date}, amount)$;

$Project(\underline{ProjNo}, ProjName)$;

Also assume we have the following inclusion constraints:

$DrivingLicense[Pno] \subseteq Person[Pno]$;

$PersonPhone[Pno] \subseteq Person[Pno]$;

$Employee[Pno] \subseteq Person[Pno]$;

$ProjStaff[Eno] \subseteq Employee[Eno]$;

$PorjStaff[ProjNo] \subseteq Project[ProjNo]$;

$SalaryHistory[Eno] \subseteq Employee[Eno]$;

By applying clustering rule, we can identify the following three clusters: Cluster Person = {Person, DrivingLicense, PersonPhone} with relation Person as the main class relation. Cluster Project = {Project} with relation Project as the main class relation. Cluster Employee = {Employee, SalaryHistory} with relation Employee as the main class relation. Relation ProjStaff will give rise to a relationship object class as discussed in section 7.

Apply ISA identification rule on relations Employee and Person, we identify the following ISA relationship:

$Employee\ ISA\ Person\ via\ Employee[Pno] \subseteq Person[Pno]$.

The following class specifications will be generated in the target schema:

```
class Person {
      string Pno;
      string name;
      int age;
      string address;
      string DrivingLicense_licenseNo;
      setof(string) PersonPhone_phoneNo;
      };

class Employee: public Person {
      string Eno;
      string Pno;
      DATE dateOfJoin;
      setof(tuple<DATE: date, integer: amount>) SalaryHistory;
      };

class Project{
      string ProjNo;
      string ProjName;
      };
```

We need to set up a well-defined inheritance mechanism. In the above example, it seems that Employee.Pno should be inherited from class Person. We still include attribute Pno in class Employee for two reasons. First, since the OODB schema is the result of translation from a relational database schema, keeping this attribute may give more space for later optimization of queries on OODB schema. Second, Pno in Employee is a "property" of Employee as well as a link that connects an Employee to a Person. In this sense, it should be kept in Employee.

Apparently, there will also be the problem of attribute name conflict. In the above example, it does not matter since the Person an Employee relates to has the same Pno value with the Employee. In general, user interference will be needed for resolving the conflicts. This will be not be further discussed in this paper.

**Discussion.** We give a brief discussion on avoiding unnecessary ISA links between object classes. In the above example, intuitively the relation DrivingLicense may give rise to an object class "Person with driving license" which is a subclass of Person. This will not be the result of our translation. Notice that no other relation references the relation DrivingLicense, i.e. the object class "Person with driving license"(if any) does not participate in any relationship with other objects. In our approach, this type of relation will be taken as fragment of object class rather than independent object class. This will help avoid proliferation of classes and hence unnecessary ISA links among classes. However, it might be desirable in user's view point that "Person with driving license" be

an independent class. This can be handled by another layer of view mechanism by which users can do more complicated restructuring of schema. For the above example, a user can get a new subclass of Person, "Person with driving license" by using *specialization*[1] on class Person.

# 6 Object Identifier Generation

In our approach, each object instance will have a unique object identifier (oid). The concept of oid is not inherent for RDM. In RDM, identification of objects is by keys. The difference between the two ways of identification is that, in OODM, oid is not value-based while in RDM, keys are basically values.

In the context of translating a relational schema to an OODB schema, objects are sort of "imaginary" in the sense that they do not exist in a relational database physically. The identification of these imaginary objects rely on that in the relational database which is value-based. In our approach, we generate oid's based on key values. This idea is similar to the generation of oids for imaginary objects by using "core attributes"[1]. We consider the following cases for oid generation.

### Object class with no superclass

Consider an object class R with main class relation R whose primary key is K. Assume that object class R has no superclass. For an instance of R with key value k, its identifier is formed by concatenating the name R with the value k. For example, for a Person with Pno = "1234", its oid will be "Person.1234". Notice that given this oid, the translator can interpret it correctly and is able to access all the relevant information of the particular Person object identified by key value "1234".

### Object class with exactly one direct superclass.

Consider a class R with superclass S. An instance of class R is also an instance of class S. Hence the oid for an instance of class R should be understandable to the translator as that for an instance of class S as well as one in class R. Consider the classes Person and Employee in Example 5.1, where class Employee is a subclass of the class Person. Assume that there exists an Employee with Eno = "456" and Pno = "1234". In the current approach, the oid for this instance will be chosen as "Person.1234" rather then "Employee.456". Notice that Pno is always a key property of Employee. This fact ensures that the oid "Person.1234" can be correctly interpreted even as oid for an Employee instance. In our implemented algorithm, this process is recursive.

In general, if an instance is contained in multiple classes due to the existence of ISA relationship, the oid will be chosen to be most general so that it can be interpreted correctly in the context of all the classes of which it is a member. Apparently, if multiple

direct superclass does not happen, this method is sufficient.

**Object classes with more than one direct superclass.**

We consider cases where there exist multiple direct superclasses. Assume we have class C with direct superclasses $C_1$ and $C_2$. If $C_1$ and $C_2$ has a common superclass, $C'$, class $C'$ can be used as the base for oid generation for class C. If this is not the case, let $C_1'$ and $C_2'$ be the most general superclasses of classes $C_1$ and $C_2$, respectively, the oid for instances of class C can be generated by concatenating the oid's of classes $C_1'$ and $C_2'$. For example, consider an object class Student-Employee which is a subclass of classes Employee and Student. If both Employee and Student classes are subclasses of class Person, the oid of Student-Employee will be in the form of "Person.1234". If Employee and Student exist as independent classes, i.e. they don't have a common superclass, the oid of Student-Employee will be in the form of "Student.0123::Employee.456", where "0123" is the a value for Student.Sno, key attribute of class Student. By doing so, we make sure that a Student-Employee oid can be correctly interpreted as a Student, an Employee or a Student-Employee.

Oid for relationship object classes can be constructed by concatenating the oids of the participating objects. This will be mentioned again in section 7. The oid in our context is conceptual rather than physical. We support persistent identifiers for object. However, these oid's will be interpreted by the translator into data values which is used to access real data in the relational database. A detailed description of this issue will be necessary if query translation and data integration is to be discussed in length. While we'll give a brief overview of the two issues in the final section, the detail is not within the scope of this paper.

# 7 Inter-Object Relationships

Inter-object relationships may exist in relational database in two forms. First, as relations whose primary key consists of disjoint foreign keys. For example, the ProjStaff relation in Example 5.1. Second, as foreign keys in relations that is contained in a relation cluster representing an object class. We discuss both cases in this section.

Intuitively, relations whose primary key consists of disjoint foreign keys will be treated as a relationship object class. Notice that a relationship object class is very similar to the object class we described earlier except the followings:

1. The primary key is nonatomic in the sense that it contains more than one disjoint foreign key. This reflects the fact that a relationship always involves more than one object class.

2. There is no ID-dependency or ISA relationship among relationship object classes. Hence the translation is easier.

We illustrate the translation by an example:

**Example 7.1.** Consider the ProjStaff relation in Example 5.1.

$ProjStaff(\underline{ProjNo, Eno}, position)$;

From the assumptions given in Example 5.1, we know that ProjNo and Eno are foreign keys referencing object classes Project and Employee, respectively. Relation ProjStaff represents a relationship object class as follows:

```
class ProjStaff {
      Project ProjNo_Project;
      Employee   Eno_Employee;
      string position;
      };
```

In general, a relationship object class can be represented by a cluster of relations similar to the case described in section 3. We will not give the details here. Object identifier for a relationship object instance can be easily generated by concatenating the oids of all the participating objects together.

The identification of foreign keys is quite simple after we have identified all the non-relationship objects. A foreign key in a member relation of a cluster representing an object class will give rise to an attribute whose value is a (set of) oid. This is illustrated by the following example.

**Example 7.2.** Consider the following relations with keys underlined:

$Student(\underline{Sno}, Grade, Dno)$;

$Department(\underline{Dno}, Dname)$;

Assume that inclusion constraint Student[Dno] $\subseteq$ Department[Dno] holds. The two relations will be both identified as main class relations leading to classes Student and Department. Notice that Student.Dno is a foreign key, we replace it by an oid valued attribute. This will give the following specification for the class Student:

```
class Student{
      string Sno;
      int Grade;
      Department Dno_Department;
      };
```

Notice that the name of an object valued attribute is suffixed with the name of its object type. This is to indicate that the attribute takes on oid's as values. This is another detail of the algorithm.

# 8 Conclusion and Perspectives

In this paper, we present an approach for schema translation from RDM to OODM. The approach extracts semantics from a relational database schema by using the key constraints and inclusion constraints. The major contribution of this work is to give a systematic and rigorous way of identifying OO semantics from a relational database. To do this, we also allow user to specify certain useful data characteristics of the relational database in the form of key constraints and inclusion constraints. The followings can be achieved automatically:

1. Identification of relation clusters representing object classes.
2. Identification of ID-dependencies between the identified object classes.
3. Identification of ISA hierarchy of object classes.
4. Generation of object identifiers.
5. Identification of inter-object relationships.

As the result, a semantically clean OODB schema can be generated. The result OODB schema can be used easily by a multidatabase view mechanism for generating views based on heterogeneous databases.

A complete translation algorithm is complicated and is yet to be completely established. The correctness of the algorithm need to be proven. A complete and formal presentation of a translation algorithm is given in [20].

As future research, we will address the issues of query translation and data integration. Query translation is the issue of translating the OO queries against target schema into relational queries that can be processed by the underlying relational database. The OO query language we choose is a subset of O2 query language [2]. Currently, we are developing a rewriting method to handle this issue. The basis of this approach is to express each relation cluster representing an object class by a formula in some sort of algebra, e.g. relational algebra extended with a "nest" operator to represent set-valued attributes. This rewriting method can be integrated with an OO query processor to transform OO query into executable relational database queries. Data integration is the issue of reorganizing the data retrieved from the relational database into proper form to be understood by users of the OO schema. Solution to the above issues will make our schema translator complete.

Another interesting issue is the updatability of the target OODB schema as the result of translation. An intuitive conjecture is that the OODB schema generated by using our translation algorithm is updatable. A formal investigation of this problem will be based on an analysis of the algebraic formula mentioned in the last paragraph. If this formula provides enough information to eliminate ambiguities(if any), the target schema will be

updatable.

**Acknowledgment.** We thank Dr.Desai Narasimhalu, Dr.Surjatini Widjojo and Dr.Anne Ngu, members of the Heterogeneous Database group at ISS, for their suggestions and support.

# References

[1] S.Abiteboul, A.Bonner, *"Objects and Views"*, SIGMOD 1991, May 1991, Denver, Colorado, USA.

[2] F.Bacilhon, S.Cluet, C.Delobel, *"A Query Language for the O2 Object-Oriented Database System"*, Tech.Report 35-89, Altair, France, 1989.

[3] M.Carey et.al, *"A Data Model and Query Language for EXODUS"*, SIGMOD 1988, June 1988, Chicago, Illinois, USA.

[4] M.A.Casanova et.al, *"Inclusion Dependencies and Their Interaction with Functional Dependencies"*, Journal of Comp. and Sys.Sc., 28, 1984.

[5] M.Castellanos and F.Saltor, *"Semantic Enrichment of Database Schemas: An Object Oriented Approach"*, First International Workshop on Interoperability in Multidatabases Systems, April 7-9, 1991. Kyoto, Japan.

[6] P.P.Chen, *"The Entity-Relationship Model: Towards a unified View of Data"*, ACM TODS, 1:1, 1976.

[7] K.H.Davis, A.K.Arora, *"Converting a relational database model into an Entity-Relationship Model"*, Entity-Relationship approach, S.T.March (ed.), 1988.

[8] S.R.Dumpala, S.K.Arora, *"Schema Translation Using the Entity-Relationship Approach"*, Entity-Relationship Approach to Information Modeling and Analysis, P.P.Chen (ed.), Elsevier Science Publisher, Amsterdam, 1983.

[9] L.A. Kalinichenko, *"Methods and Tools for equivalent Data Model Mapping Construction"*, EDBT'1990, March 1990, Venice, Italy.

[10] C.Lecluse, P.Richard, F.Velez, *"O2, an Object Oriented Data Model"*, SIGMOD 1988, June 1988, Chicago, Illinois, USA.

[11] H.Mannila, K.J.Raiha, *"Inclusion Dependencies in Database Design"*, Proc.2nd ICDE, 1986.

[12] R.Missaoui, R.Godin, *"The Implication Problem for Inclusion Dependencies: a Graphical Approach"*, SIGMOD Record 19:1, 1990.

[13] J.C.Mitchell, *"Inference Rules for Functional and Inclusion Dependencies"*, Proc.ACM PODS, 1983.

[14] S.B.Navathe, A.M.Awong, *"Abstracting Relational and Hierarchical Database with a Semantic Data Model"*, Entity-Relationship approach, S.T.March (ed.), 1988.

[15] A.P.Sheth, J.A.Larson, *"Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases"*, ACM Computing Surveys, vol.22, no.3, Sept.1990.

[16] D.D.Straube, M.T.Ozsu, *"Execution Plan Generation for an Object-Oriented date model"*, DOOD, 1991.

[17] D.C.Tsichritzis, F.H.Lockovsky, *Data Models*, Prentice-Hall, NJ, 1982.

[18] G.Wiorkowski, D.Kull, *DB2: Design and Development Guide*, 2nd Edition, Addision-Wesley, 1990.

[19] E.Wong, R.Katz, *"Logical Design and Schema Conversion for Relational and DBTG Databases"*, Entity-Relationship Approach to System Analysis and Design, P.P.Chen (ed.), Elsevier Science Publishers, Amsterdam, 1980.

[20] L-L.Yan, T-W.Ling, *"Translating Relational Schema into OODB Schema Using Constraints"*, To appear as technical report in the Institute of Systems Science, National University of Singapore, 1992.