Object View Derivation and Object Query Transformation*

Chih-Chin Liu and **Arbee L.P. Chen**

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
Email : alpchen@cs.nthu.edu.tw

Abstract

In this paper, we propose an approach for building an object front-end on top of a relational database. The object front-end can be used as the interface for integration with other object databases. The users are allowed to access data stored in the relational database using an object query language. A set of derivation rules is provided to construct an object view from the relational schema. Transformation strategies from object queries to corresponding relational queries are also presented. This object front-end has been implemented in a heterogeneous database project.

1 Introduction

It is getting common that many databases with distinct data models co-exist in a distributed environment, the need to integrate relational and object database systems for data sharing becomes more and more essential. We have started a project for this purpose, in which an object front-end is constructed on top of a relational system and integrated with other object database systems [8] [3] [4]. Many other approaches have been proposed. On one hand, multidatabase systems which provide users a powerful query language to access local databases have been proposed in [9] [10]. However, users lack a global view to help them retrieve the data scattering in local databases.

On the other hand, federated database systems [7] [13] provide users an integrated global view of data at each local site. Schema integration is the main consideration of federated database systems, Batini et. al. [2] provides a comparative analysis. To deal with the complexity of the task, we usually assume that all schemas to be integrated are homogeneous, i.e. of the same data model. Hence, a canonical data model is needed. Due to its rich type system and its ability of capturing more semantics, object data model is frequently chosen as the canonical one [12]. If the data model of a local schema is different from the canonical one, schematic conflicts occur. There are two approaches to resolve the schematic conflicts. One approach is adding the features of the canonical data model to the local database systems [6] [1]. Polyglot [6] adds the object features to an existing relational database system. Ananthanarayanan et. al. [1] proposes the co-existence approach which allows the relational database system to access the objects generated by an object applications. However, this approach may break the autonomy of local database systems. Moreover, the interface is designed for object applications, not for object database systems.

The other approach to resolve schematic conflicts is schema transformation [11] [15]. Transforming an object schema into a corresponding relational schema has been proposed in [11]. Since the object data model is more powerful than the relational data model, the mapping goes straight forward. Yang and Ling [15] proposes some rules to transform a relational schema into its object counterpart. However, according to their rules, the complex set attributes of classes cannot be constructed, and the derivation of class hierarchies using common attributes is also not available. The transformation from object queries into the corresponding relational queries was not discussed either.

In this paper, we propose a set of derivation rules which can construct an object view on top of a relational database. These rules state the ways to find the object features such as inheritance and complex objects according to the schema and constraints of a relational database. Bases on these derivation rules, we also discuss how to transform an object query into the corresponding relational query. Our approach can be thought of as building an object front-end on top of an existing relational database. Thus the autonomy of the local database systems is preserved.

The rest of this paper is organized as follows. Section 2 presents an overview of the object front-end construction. The derivation rules for constructing an object view are provided in section 3. The object query transformation strategies are proposed in section 4. Section 5 discusses some issues of query result transformation. Finally, section 6 concludes this paper.

*This work was partially supported by the Republic of China National Science Council under Contract No. NSC 83-0408-E-007-030.

2 Overview of Object Front-end Construction

In a heterogeneous database system, there may exist many autonomous database systems with different data models. To provide users a single query language to retrieve data from these databases, an integrated global schema should be developed from the component schemas. Due to the expressive power and its rich type system, object data model is adopted frequently as the canonical model for database integration.

We consider a heterogeneous database system which contains two kinds of database systems: the relational(Sybase and Informix) and object(SQL/X [14]) database systems. Since a global object schema is integrated from object schemas of the component database systems, we have to construct an object front-end for each component relational database system.

Besides the usage for integration, the object front-end also enables a relational database system to act like an object database system. It serves two functions. One is to provide users an object view for posing queries in an object query language. The other is to transform an object query into its corresponding relational query and to transform the query result from the relational database system into the corresponding object format. The architecture of a relational database system with object front-end is shown in Fig. 1 . Note that the object view statements are provided by the DBA. However, in the following of this paper, we will show that a semi-automatic derivation of object views is possible.

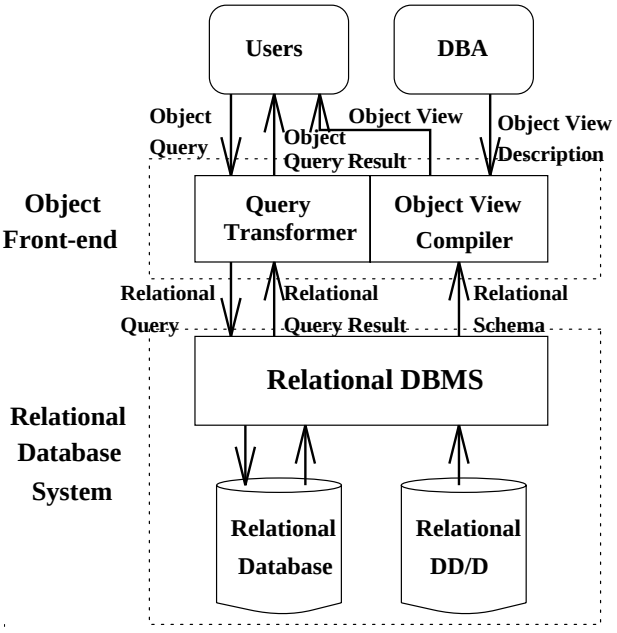

Figure 1: Architecture of an Object Front-end.

3 Object View Construction

Before we discuss the derivation rules for the object views, we briefly review the important features of object databases and compare the relational data model with the object data model.

3.1 The Comparison of Data Models

The comparison of data models consists of two parts. First, we analyze how a real world entity is stored in these two models. Then, we discuss the ability of capturing relationships in these two models.

A. Entities: Object is the most primitive notion in the object techniques. An object represents a real world entity and is assigned by system a unique value call object identifier(OID). An object may have several attributes. Unlike the relational data model in which only primitive attributes are allowed, attributes in object data models can be primitive, complex, set(multi-valued), or complex set.

Encapsulation is another important notion of the object data model. The properties and the behaviors of an object are encapsulated in the object. There is no encapsulation notion in the relational data model, that is, a relation has properties but no behaviors.

A relational database consists of a set of relations. Roughly speaking, a relation corresponds to a class. A relation consists of a set of tuples which corresponds to an object. The relation name and its key can be combined to identify any tuple within a relational database.

An attribute may be multi-valued in the real world. When we design a relational schema, a relation with set attributes is usually divided into several relations. So the semantics of the set attributes is hidden in the relational schema.

B. Relationships: The definition of relationships is explicit in the object-oriented database. Basically, there are two kinds of relationships supported in the object data model: **generalization(IS-A)** relationship and **aggregation** relationship. If a class A inherits from another class B, we say A is a subclass of B or B is a generalization of A. There are two types of inheritance model in an object database. If a superclass contains all the objects in the subclasses, we call the inheritance model a **dynamic model**. On the other hand, if a superclass does not contain any object in the subclasses, we call it a **static model**. Our object view derivation is based on the static inheritance model.

There is no inheritance concept in the relational data model. Depending on the database designers, the schema for two entities with the IS-A relationship can be quite different. Thus we have many approaches to derive the IS-A relationship from a relational database.

Aggregation relationships enable the object database system to model complex objects more naturally. If the value of an attribute of an object is the OID of certain object, then there is a aggregation relationship between the two objects. This is similar to the referential integrity constrain in the relational database. The primary key/foreign key links act like domain/attribute links.

We conclude the above discussion in the table shown in Table. 1.

Relational Data Model	Object Data Model
relation	class
tuple	object
relation name + key	OID
attribute	primitive attribute
referential integrity	complex attribute
additional relations + constrains	set attribute
additional relations + constrains	complex set attribute
none	encapsulation
none	method
none	inheritance

Table 1: The comparison of data models.

3.2 Derivation Rules

3.2.1 Inheritance Relationship

There are two ways to derive the inheritance relationship from a relational schema. One is based on the key containment relationship, the other the common attributes of the two relations.

A. Derive from Key Containment Relationship

Rule 1 *If* Π_r $R \supset \Pi_s$ S *then* S *is a subclass of* R*, where* R*,* S *are relations, and* r*,* s *are the keys of relation* R *and* S*, respectively.*

Rule 1 says that relation R stores all the objects, while some objects may have other attributes and are stored in relation S. We can therefore derive that S is a subclass of R.

Example 1 *Assume relation Student keeps the information of all students, whose key is S_NO, and relation Graduate keeps the advisor and dissertation information of all graduate students whose key is also S_NO.*

Student(S_NO, Name, Department, Class, Address, Telephone)

Graduate(S_NO, Advisor, Dissertation)

If Π_{S_NO} *(Student)* $\supset$ Π_{S_NO} *(Graduate) then Graduate is a subclass of Student.*

B. Derive from Common Attributes

Rule 2 *If* $A(R) \cap A(S) = T \neq \phi$ *then we can derive a superclass of* R *and* S *with* T *as its attribute sets , where* $A(R)$ *and* $A(S)$ *denote the attribute sets of* R *and* S*, respectively.*

Rule 3 *If* $A(R) \supset A(S)$ *then relation* S *is a superclass of relation* R *,where* $A(R)$ *and* $A(S)$ *denote the attribute sets of* R *and* S*, respectively.*

Example 2 *Consider the following relational schema:*

Graduate(S_NO, Name, Department, Advisor, Major, Dissertation)

Undergraduate(S_NO, Name, Department, GPA)

Since A*(Graduate)* $\cap$ A*(Undergraduate)* $=$ {*S_NO, Name, Department*}*,*

according to rule 2, we can derive a superclass Student(S_NO, Name, Department) of Graduate and Undergraduate.

Example 3 *Consider the following relational schema:*

Student(S_NO, Name, Department)

Graduate(S_NO, Name, Department, Advisor, Major, Dissertation)

Since {*S_NO, Name, Department*} $\subset$ {*S_NO, Name, Department, Advisor, Major, Dissertation*}*,*

according to rule 3, we can derive that Student is a superclass of Graduate.

3.2.2 Complex Attributes

Rule 4 *Assume* R *and* S *are two relations. If* r *is a foreign key of* R *with respect to* S*, then* S *is the domain of* r*.*

Rule 4 says that r is a complex attribute of relation R. We can put the value of the complex attribute in R. However, to save the storage space, each distinct value of the complex attribute is stored in another relation. In relation R, we only store the foreign key values. By using join operation, we can get all the values of the complex attribute.

Example 4 *Consider the following relational schema:*

Student(S_NO, Name, Department, Class, Advisor)

Teacher(T_NO, Name, Department),

where T_NO is the key of Teacher, and Advisor is a foreign key of Student with respect to Teacher.

According to rule 4, we derive that Teacher is the domain of attribute Advisor.

3.2.3 Set Attributes

Rule 5 *If* $K(R) \subset K(S)$*, and* $K(S)$ *-* $K(R)$ *does not contain any foreign key, then* S *is a set attribute of* R*, where* $K(S)$*,* $K(R)$ *are the keys of relation* S *and* R*, respectively.*

In object model, an attribute can be multi-valued, while in relational model, each attribute must be atomic. However, we can use an additional relation to store the values of the multi-valued attribute. The key of the additional relation is composed of the key of the original relation and the multi-valued attribute.

Example 5 *Consider the following relational schema:*

Student(S_NO, Name, Department, Class)

Student_Hobby(S_NO, Hobby_Name),

where S_NO is the key of Student, and S_NO + Hobby_Name is the key of Student_Hobby.

Since {*S_NO, Hobby_Name*} $\supset$ {*S_NO*}*,*

according to rule 5, we derive that Student_Hobby is a set attribute of Student.

3.2.4 Complex Set Attributes

Rule 6 *If* $K(S) = K(R) \cup K(T)$ *then* S *is a complex set attribute of relation* R*, where* $K(R)$*,* $K(S)$*, and* $K(T)$ *are keys of relation* R*,* S*, and* T*, respectively.*

A complex set attribute is an attribute that is complex and multi-valued. The key of the relation that stores the values of the complex set attribute is composed of the key of the original relation and the key of the domain.

Example 6 *Consider the following relational schema:*

Student(S_NO, Name, Department, Class)

Course(C_NO, Teacher, Room)

Student_Study(S_NO, C_NO)

where S_NO, C_NO, and S_NO + C_NO are the keys of Student, Course, and Student_Study, respectively.

According to rule 6, we derive that Student_Study is a complex set attribute of Student.

3.3 Object View Derivation Language

We define the syntax of an object view derivation language(OVDL) and show its usage with an example in the following.

The OVDL consists of five kinds of statements which describe the relationships generated by the derivation rules.

Generalization Statement:

[relation_name] **superclass_of** *[relation_name]* {,*[relation_name]*} **static** | **dynamic**

Specialization Statement:

[relation_name] {,*[relation_name]*} **superclass_of** *[relation_name]* **static** | **dynamic**

Generalization statement and specialization statement both describe the inheritance relationship derived according to rule 1, rule 2, or rule 3. Generalization statement captures the superclass/class relationship between two relations while specialization statement captures the class/subclass relationship between two relations. (The keywords static and dynamic mean that the inheritant model is based on static or dynamic model, respectively.)

Complex_Attribute Statement:

[attribute_name] **comp_attr_of** *[relation_name]* **domain** *[relation_name]*

Complex_Attribute statement specifies a complex attribute and its domain which is derived from rule 4.

Set_Attribute Statement:

[relation_name] **set_attr_of** *[relation_name]* **name** *[attribute_name]*

Set_Attribute statement specifies a relation as a set attribute of another relation according to rule 5. The name of the derived set attribute can be changed.

Complex_Set_Attribute Statement:

[relation_name] **comp_set_attr_of** *[relation_name]* **name** *[attribute_name]* **domain** *[relation_name]*

Complex_Set_Attribute statement specifies a complex set attribute with its domain and the related relation according to rule 6. Again, the derived attribute can be changed.

Example 7 *Assume there is a relational database with schema as follows:*

Student(SNO, Name, Age, Residence, Department)

Teacher(ENO, Name, Age, Residence, Salary)

Graduate(SNO, Advisor)

Undergraduate(SNO, GPA)

Assistant(SNO, ENO, Age, Residence, Department, Salary, Assist)

Address(ANO, City, Street, Zip)

Course(CNO, CName, ClassRoom)

Room(RNO, Building, Floor)

Student_Hobby(SNO, HName)

Student_Study(SNO, CNO)

The database has the following constraints and attribute set relationships:

1. Π_{SNO} *(Student)* $\supset$ Π_{SNO} *(Graduate)*

2. Π_{SNO} *(Student)* $\supset$ Π_{SNO} *(Undergraduate)*

3. $\{SNO, HName\} \supset \{SNO\}$

4. $\{SNO, CNO\} = \{SNO\} \cup \{CNO\}$

5. $\{SNO, Name, Age, Residence, Department\} \supset \{Name, Age, Residence\}$

6. $\{ENO, Name, Age, Residence, Salary\} \supset \{Name, Age, Residence\}$

7. $\{SNO, ENO, Name, Age, Residence, Department, Salary, Assist\} \supset \{SNO, Name, Age, Residence, Department\}$

8. $\{SNO, ENO, Name, Age, Residence, Department, Salary, Assist\} \supset \{ENO, Name, Age, Residence, Salary\}$

According to the object view derivation rules, we can write the following OVDL statements:

Graduate **subclass_of** *Student* **dynamic**

Undergraduate **subclass_of** *Student* **dynamic**

Person **superclass_of** *Student, Teacher* **static**

Student **superclass_of** *Assistant* **static**

Teacher **superclass_of** *Assistant* **static**

Residence **comp_attr_of** *Student* **domain** *Address*

Residence **comp_attr_of** *Teacher* **domain** *Address*

Residence **comp_attr_of** *Assistant* **domain** *Address*

Advisor **comp_attr_of** *Graduate* **domain** *Teacher*

Assist **comp_attr_of** *Assistant* **domain** *Course*

ClassRoom **comp_attr_of** *Course* **domain** *Room*

Student_Hobby **set_attr_of** *Student* **name** *Hobby*

Student_Study **comp_set_attr_of** *Student* **name** *Study* **domain** *Course*

According to these statements, the corresponding object view is constructed as shown in Fig. 2:

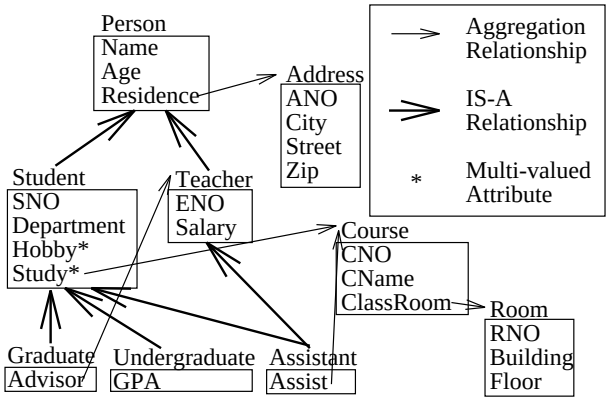

Figure 2: Object View of Example 7.

4 Object Query Transformation

4.1 Overview of Transformation

When we transform an object query into its corresponding relational query, the tasks we have to do are:

- computing the population of each class involved in the object query
- when there exist target attributes which cannot be retrieved directly from the corresponding relation of the target class(due to rule 1 derivation), joining the corresponding relation with the relation which contains the target attributes
- converting path expressions by joining the corresponding relations
- retrieving the value of each set attribute by joining the corresponding relations
- converting set operations which are applied on the set attributes into their relational counterparts

Since we adopt SQL as the target relational query language, we have to generate the SELECT, FROM and WHERE clauses.

In the following we present the transformation strategies. In each case we explain it with an example.

4.2 Query Transformation Examples

4.2.1 Simple Query

A simple query is the one which does not involve any relationship. In other words, the range of the query is within a single class. In transforming a simple query, we simply apply set operations on the corresponding relations to compute the population of the target class. If the target class is mapped directly from a relation, its population is composed of all tuples of the relation.

Q 1 *SQL/X: select X.Name from Teacher X*
SQL: select X.Name from Teacher X

Note that due to the similarity of the syntax of SQL/X and SQL, the two queries are same. However, if the target class of an object query is a superclass derived from rule 1, its population only consists of the tuples whose keys do not appear in tuples of any subclass in the derivation. We have to check the membership of each key value to decide the population.

If the target class of an object query is a superclass derived from rule 2, it does not contain any object. There is no relational query corresponding to the object query.

Q 2 *SQL/X: select X.Name from Person X*
SQL: (null)

If the target class of an object query is a subclass derived from rule 1, we have to join the corresponding relation with its superclass in the derivation to get all its attribute values.

Q 3 *SQL/X: select X.Name from Graduate X*
SQL: select X.Name
from Graduate X, Student Y
where X.SNO = Y.SNO

Since the names of all graduate students are stored in the Student relation, we join Graduate with Student to get them.

4.2.2 Query with Complex Attributes

The path expressions provide users a convenient way to express queries with complex attributes. Since every aggregation relationship is derived from the referential integrity constrain of the relational database, we join the target attribute relation with the domain relation and add the domain relation to the FROM clause. If a path expression relates to n aggregation relationships, then we need n join operations to implement the transformation. The path expressions may appear in the SELECT clause or the WHERE clause of an OO query.

Q 4 *SQL/X: select X.Name, X.Residence.City*
from Teacher
SQL: select Teacher.Name, Address.City
from Teacher, Address
where Teacher.Residence = Address.ANO

4.2.3 Query with Set Attributes

Since a set attribute is derived from rule 5 which enforces a constrain $K(R) \subset K(S)$, we join the key of S with the corresponding foreign key of R for the transformation. However, if there exist one or more target attributes which are multi-valued, the result of the object query is not in first normal form. It means we have no exactly equivalent relational query in this situation. We can apply an additional operation **nest** to transform a relation into non-first normal form. This will be explained in the next section.

Q 5 *SQL/X: select X.Name, X.Hobby*
from all Student X
SQL: select Student.Name,
Student_Hobby.HName
from Student, Student_Hobby
where Student.SNO = Student_Hobby.SNO
nest(Student_Hobby.HName)

4.2.4 Query in Class Hierarchies

In an object database, we can specify the range of a query as the whole class hierarchy or some classes in the class hierarchy. A direct transformation of this kind of queries is to compute the population of each class included in the range and then union these populations. Obviously, this approach can be improved by combining some subqueries into one subquery and removing the redundant subqueries.

Q 6 *SQL/X: select X.Name from all Person X*
SQL: select Student.Name from Student
union select Teacher.Name
from Teacher
union select Assistant.Name
from Assistant

5 Query Result Transformation

We have proposed transformation strategies that can translate an object query into its corresponding relational query. However, due to the difference between the object and relational data models, there exist parts of object queries which cannot be directly transformed . We need additional operations on the relational query results to make them appear as the object query results.

5.1 The Generation of Object Identifiers

In an object database, every object has a unique system-assigned value called OID. In a relational database the relation name and its key can be used to identify any tuple. However, some relations are designed to store the values of set attributes or the complex set attributes. Since these relations do not represent weak entity sets[5], we need not generate OIDs for them.

There are several approaches to generate the OIDs. One is to build a mapping table with three attributes: relation_name, key_value, and OID. When we build an object view, we assign each tuple a unique value and insert it with the associated key value and relation name into the OID mapping table. A drawback of this approach is that we need a large space to store the OID mapping table. Also, the length of keys may diff in two relations.

Another approach of the OID generation is by using hash functions. The relation name and key value are used as the parameters of a special OID hash function which returns an OID. However, hash conflict is a main problem which needs to be solved.

5.2 The Composition of Set Attribute Values

In a relational database, all attributes are single-valued. While in an object database, there may exist multi-valued attributes. Since a relational query does not have the ability to generate multi-valued results of some set attributes, we have to combine several tuples of the relational query result to compose a set value. We define nest(A) which transforms the tuples in a relation into objects with a multi-valued attribute A whose value is the set of different values of attribute A in the object.

Note that the nest operation operates on the relational query result. When an object query is transformed into an SQL query, we add this operation at end of the SQL query if the target attributes contains a multi-valued attribute.

6 Conclusion

In this paper, we present an approach to build an object front-end on top of an existing relational database. The construction of an object front-end is achieved in two steps. First we derive the object view from the underlying relational schema. Then we build the query translator.

We have defined a set of derivation rules which can derive the following object features:

- class hierarchies
- complex attributes
- set attributes
- complex set attributes

To specify the derivation rules, we defined a derivation language called OVDL. The DBA can write OVDL statements to build an object view. However, since the constrains of the derivation rules are available from the DD/D of the underlying relational database, the OVDL statements can be generated automatically by the system. These automatically generated OVDL statements may need to be checked by the DBA to make sure they really have the semantics as they imply.

We have also proposed the transformation strategies of the object queries based on the object view into the corresponding relational queries. The algorithm for the query transformation is not described formally in this paper. Our future work is to develop a complete and formal presentation of a query transformation algorithm considering all object features. Another important issue to consider is the optimization of query transformation. It is somewhat different from the query optimization in relational database systems since the former needs the information stored in the object view. These issues need to be studied for an implementation of an efficient query translator.

References

[1] R.Anathanarayanan et. al., "Using the Co-existence Approach to Achieve Combined Functionality of Object-Oriented and Relational Systems", Proc. of ACM SIGMOD Conf, Washington, DC, USA, 1993, pp.109-118

[2] C. Batini et al., "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computing Surveys, Vol.18, No.4, 1986.

[3] A.L.P. Chen, P.S.M. Tsai, J.L. Koh, "Identifying Object Isomerism in Multidatabase Systems" (submitted for publication).

[4] A.L.P. Chen, J.L. Koh, T.C.T. Kuo, C.C. Liu, "Schema Integration and Query Processing for Multiple Object Databases" (submitted for publication).

[5] P.P. Chen, "The Entity-Relationship Model - Towards a Unified View of Data", ACM Tran. on Database Systems, Vol.1, No.1, March, 1776, pp.9-36

[6] L. G. DeMichiel et. al., "Polyglot: Extensions to Relational Databases for Sharable Types and Functions in a Multi-Language Environment", Proc. of Ninth IEEE Int. Conf. on Data Engineering, Vienna, Austria, 1993.

[7] D. Heimbigner and D. McLeod, "A Federated Architecture for Information Management", ACM Trans. Off. Inf. Syst., Vol.3, No.3, July, 1985, pp.253-278

[8] J.L. Koh and A.L.P. Chen, "Integration of Heterogeneous Object Schemas," Proc. Entity-Relationship Approaches, 1993.

[9] W. Litwin et. al., "MSQL: A Multidatabase Language", Information Science, Vol.48, No.2, July, 1989

[10] W. Litwin, L. Mark, and N. Roussopoulos, "Interoperability of Multiple Autonomous Databases", ACM Computing Surveys, Vol.22, No.3, September, 1990, pp.267-293

[11] W. Meng et. al., "Construction of a Relational Front-end for Object-Oriented Database Systems", Proc. of Ninth IEEE Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp.476-483

[12] F. Saltor, M. Castellanos, and M. Garcia-Solaco, "Suitability of Data Models as Canonical Models for Federated Databases" ACM SIGMOD RECORD, Vol.20, No.4, December, 1991, pp.44-48

[13] A. P. Sheth and J. A. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, Vol.22, No.3, September, 1990, pp.183-236

[14] UniSQL, Inc., "UniSQL/X Database System User's Manual", Release 2.0, Austin, Texas, 1993

[15] L. L. Yan and T. W. Ling, "Translating Relational Schema With Constraints Into OODB Schema", Proc. of IFIP Conf. on Interoperable Database Systems, Australia, November, 1992, pp.69-85