Object-Oriented Modeling of Meta Information for Semantic Schema Enrichment and (Semi-)Automatic Schema Transformation

Daniel A. Keim, Hans-Peter Kriegel, Andreas Miethsam

Ludwig-Maximilians-Universität München
Institut für Informatik
Bericht 9306
April 1993

Object-Oriented Modeling of Meta Information for Semantic Schema Enrichment and (Semi-)Automatic Schema Transformation

Daniel A. Keim, Hans-Peter Kriegel, Andreas Miethsam
Institute for Computer Science, University of Munich,
Leopoldstr. 11B, D-8000 Munich 40
{keim, kriegel, miethsam}@dbs.informatik.uni-muenchen.de

Abstract

In this paper, we describe a framework for an object-oriented modeling of meta information. The meta information consists of all kind of information necessary to access and interoperate the participating databases. As part of the meta information, we model the common properties and differences of the various data models and concrete systems. We also include information to semantically enhance the schemas of the participating databases and provide the basis for an automatization of the schema transformation process. We describe possibilities to (semi-)automatically transform flat relational schemas into structured object-oriented ones which are more adequate to model complex real world problems. We use the object-oriented data model as target model of the transformation because of its semantic richness and its availability as commercial systems. The meta information necessary for the schema transformation may be user supplied or it may be deduced from an entity-relationship design schema and corresponding mapping information.

Keywords: *interoperability, federated databases, global conceptual schema, schema transformation, schema enrichment, translation of relational operations*

1. Introduction

In a federated or multi-database system there is a need to store information on the participating databases. This so-called ‘meta information’ includes name and location (network-server-address), information on the data model, access rights and so on. In today’s database systems the meta information is usually kept in some kind of data dictionary. It is easy to store the meta information in the data dictionary as long as we are only dealing with a system using a single data model. In the relational model, for example, there is only information on tables with their attributes and corresponding domains in the data dictionary. All meta-information on tables has the same structure and therefore it is adequate to use a standard data dictionary. In a federated database environment, however, the meta information is structured differently in the various databases to be integrated. Obviously, all database systems using the same data model have common structural properties of the meta information but database systems using different data models use meta information of different structure. When considering concrete (commercial) database systems using one data model e.g. relational database systems, such as Oracle and Ingres, the same situation may occur. All Ingres databases have a set of common properties but these properties might differ from the ones of Oracle databases.

To adequately model meta information, we use an object-oriented class hierarchy for the participating databases. The information on common properties of the participating databases and differences between them is represented by class attributes or member variables. We extend the meta information to also include information necessary for an interoperation of the participating databases. Since our goal is to achieve an interoperation based on logical data integration rather than physical data exchange we provide a basis for a semantic enrichment of the schema information available in the participating databases. This additional semantic information is essential to support an automatic schema transformation from the original data model to the integrating common data model.

The choice of the common data model and data manipulation language is crucial since it must provide adequate concepts to model the semantic information and integration mappings for all databases participating in the federation. Only semantic data models [HK 88] such as the functional model [Shi 81] [LR 82], the extended entity-relationship model [TYF 86] [DA 87] [NA 87] and the object-oriented model [Kim 90] [KDN 90] [CT 91] are candidates providing the needed data modeling capabilities. In our approach, we use the object-oriented data model as common data model because of its semantic richness and availability as research prototypes and commercial object-oriented database systems which have been built over the last decade and are ready to be used now. We prefer the object-oriented model over the functional data model because, to our opinion, the object-oriented model is closer to the users view of the real world modeled in the database. We also believe that the object-oriented data model is better suited as common data model than the entity-relationship model because in the object-oriented model only one concept (objects) is used as opposed to the two concepts (entities and relationships) of the entity-relationship model. Using two concepts may cause problems in the process of schema integration and schema translation because the same real world object may be modeled as entity in one schema but as relationship in another schema. In the object-oriented data model everything is modeled as object and therefore it is easier to integrate different schemas. An example for a project using the object-oriented model as common data model is the Pegasus project [Ahm 91] at the Hewlett-Packard Research Laboratory in Palo Alto. In Pegasus, for each relation automatically a class with member variables for all attributes of the relation is created which may be accessed like any other class in the object-oriented database. Although using the semantically rich object-oriented data model, the created structure of the schema remains flat as it is in the relational model. In order to (semi-) automatically create more structured classes from relational schemas, in our approach, we use the object-oriented meta information for semantic schema enrichment.

There are many approaches to enrich existing data models with more semantics, most of which are independent of using the additional semantics for an interoperation of heterogeneous databases. Codd, for example, proposed an extended relational data model [Cod 79] and a lot of research has been done in the area of NF^2 relational systems but only few of the proposed extensions to the relational data model have been integrated into existing systems so far. Also research has been done to extend the entity-relationship model [TYF 86] [Mar 87] and to semi-automatically translate relational schemas into extended entity-relationship schemas [DA 87] [MM 90]. Much research has been carried out in this area and many extensions for the entity-relationship model have been proposed, but, to our knowledge, none of these extensions resulted in a working system so far. In the context of schema translation for interoperation of databases also a semantic extension of the object-oriented model has been proposed [CS 91]. The extensions are used to model special types of relationships between objects such as cover aggregation or existence dependency. In contrast to the work of Castellanos and Saltor, we use the object-oriented model as provided by commercially available systems without adding any special extensions [Ita 91].

Another approach to schema enrichment for interoperating databases is the Carnot project [CHS 91]. In Carnot, a large knowledge base is used to automatically provide the information necessary to access and interoperate the heterogeneous databases participating in the federation. We believe that the Carnot approach will work nicely for some application domains with clear naming conventions and simple relationships, but for other application domains with fuzzy naming and complex relationships user, interaction is necessary to get the needed meta-information. It is impossible to guarantee that a knowledge base covers knowledge about special application domains. Furthermore, even simple attribute names or relationships cannot be automatically decoded. In our approach, we therefore demand user interaction for the semantic enrichment process which is used as a basis for the automatic schema transformation.

The paper is organized as follows: Chapter 2 introduces the object-oriented modeling of meta-information as a basis to access and interoperate the databases and to enrich the semantic information available in their schemas. Chapter 3 elaborates on the possibilities to (semi-)automatically transform relational schemas into object-oriented ones. It also includes a short description of inter-database relationships which are necessary in a multi database environment. In chapter 4, we summarize our approach and point out some problems related to an automatic translation of data manipulation operations in a multi database environment.

2. Modeling the Databases Participating in the Federated Database

As already mentioned, we propose an object-oriented modeling of the databases participating in the federated database system. The class hierarchy is presented in Figure 1.In the hierarchy the class ‘Database’ has subclasses representing ‘Relational DB’, ‘Network DB’, ‘Hierarchical DB’ and ‘Object-oriented DB’ and, since files may be considered as databases as well, a subclass called ‘Files’. Each of these classes divides into subclasses according to the concrete (commercial) systems available. The classes model the differences between the concrete database systems explicitly, e.g. the differences in specifying the location of a database. Each of the classes corresponding to one data model has a subclass called ‘Result DB’ which is used to handle results obtained from queries to different concrete systems using the same data model. Let us assume that we pose a query involving several relational systems such as Oracle, Ingres and others. The result of such a query is not limited to a single of the participating concrete systems (i.e. Oracle or Ingres), the system, however, should be able to operate on such tables

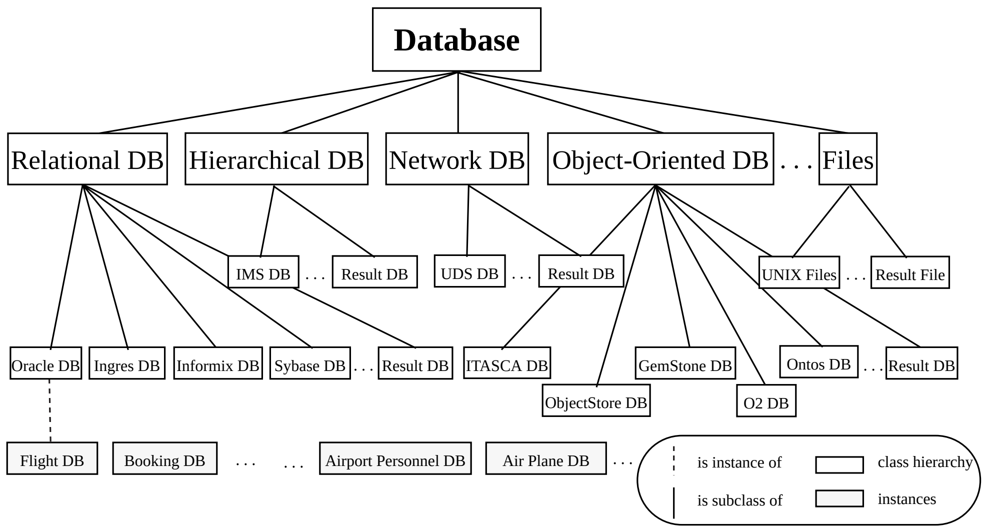

Figure 1: Object-Oriented Modeling of Federated Databases

as well. Using the subclass mechanism is a good approach since the class ‘Result DB’ inherits all properties specific to the corresponding data model (i.e. the relational model) but has no information related to a concrete database system.

The properties of classes are modeled using class attributes or member variables as they would be called in object-oriented database systems based on C++. The properties common to all classes of the hierarchy are modeled in the class ‘Database’, the properties common to all databases of one of the supported data models are described in the five subclasses and properties common to concrete (commercial) databases are handled in the corresponding classes. In Figure 1 and Figure 1 only a small part of the hierarchy is depicted in detail but it gives an impression how meta-information is represented in our system. Member variables such as ‘number_of_records’ or ‘db_size’ may be used for the query optimization process. Because of autonomy requirements, however, they might not be available for all databases and tables.The member variable ‘description’ contains a natural language description of the database and its contents. This information is useful for the user in determining the desired databases and in the future, it may also be

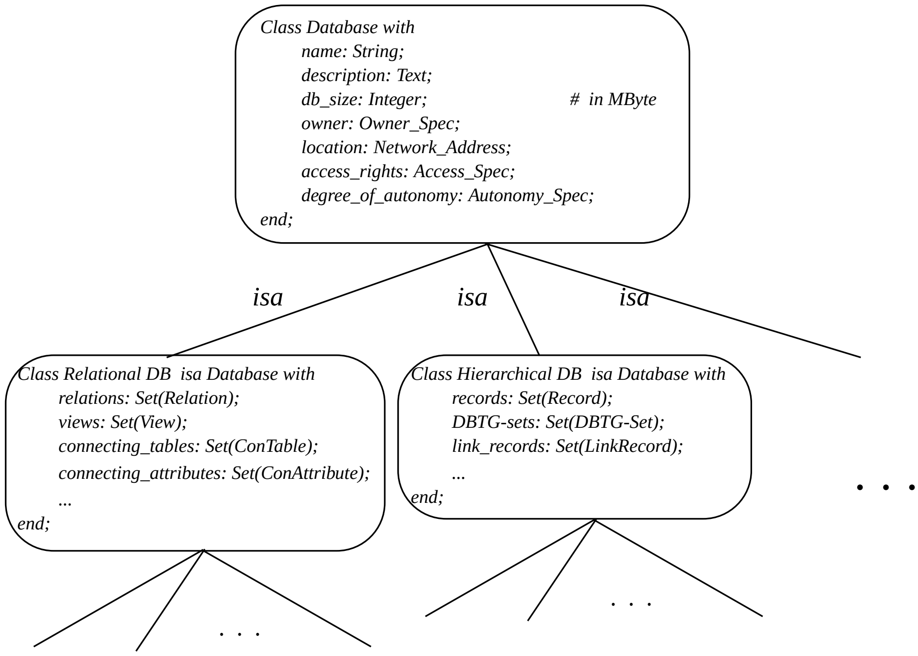

Figure 2: Object-oriented Modeling of the Federated

used as a basis for a content oriented search as done in DEMOM [HLR 90].

Special member variables are used to enhance the semantic information available in the data model or concrete system. An example in the relational model is the member variable ‘connecting_tables’ (see Figure 1) used for information on tables implementing a m:n, 1:n or 1:1 relationship. This information is useful not only for adequate access to the databases but also for the schema transformation process. The ‘connecting_tables’ indicate tables which may not be transformed into objects. Another example is the member variable ‘connecting_attributes’ which contains information on attributes used to join tables. This member variable contains semantic information on relationships between tables and on corresponding attributes which are not modeled explicitly in the relational model. The additional semantic information is necessary to automatize schema translation and integration of the database schemas using different data models and database systems. It is used to enable the interoperation of the databases and to support the access of several databases with one query interface. The (semi-)automatic schema translation process from the relational to the object-oriented model will be described in the next chapter.

At this point it should be mentioned that the meta information on the available databases and their properties such as location, degree of autonomy or access rights can be changed dynamically. This is important because the mentioned parameters may change very often for some of the participating databases. In case of a change the system has to ensure that accesses to the databases do not conflict with the new situation. This can be achieved using the 'transformed _from' links between federated schema and meta information which will be described in the next chapter.

```
Class Relation with
  name: String;
  description: Text;
  attributes: Attribute;
  number_of_records: Integer;
end;

Class ConTable isa Relation with
  connected_tables: Set(Relation);
  connecting_attributes: Set(ConAttribute);
end;
```

Figure 3: Modeling of Relations and Connecting Tables

For all classes also member functions (methods) are defined. These member functions model the query languages available for the databases of the corresponding class. For the class 'Relational DB' there are functions supporting standard SQL. The subclasses representing concrete database systems such as Oracle or Ingres contain specializations and additional functions supporting the special functionality of the concrete system.

Our class hierarchy is independent of the concrete databases to be integrated and also independent of the federated database to be created. Therefore, our modeling of databases as object-oriented class hierarchy has to be performed only once. The instantiation of the classes with concrete databases has to be carried out for each database to be integrated. The databases which are instances of the object-oriented class hierarchy then might be used to create a federated schema. As we will describe in the next section the user is able to choose the databases he wants to integrate into the federated schema. To support the user in determining the desired databases, he can access the meta information on the available databases. As an example for a concrete instantiation of the class hierarchy let us consider instances such as Flight DB, Booking DB, Airport Personnel DB and Air Planes DB as shown in Figure 1.

Assume, a new database modeled in a database system which is not yet known to the system shall be integrated into the federation. Then, a new subclass of the class corresponding to the data model used has to be created. In the new class all information on special properties of the system has to be modeled including special semantic information necessary to integrate the database schemas with schemas of databases belonging to classes representing other database models and systems.

3. Transformation of Relational into Object-Oriented Schemas

In this chapter, we investigate how the schema information of a relational database can be transformed into class definitions in an object-oriented model. Starting from a naive approach with one-to-one correspondence of tables and classes, we will present a more elaborate approach where the object-oriented class hierarchy provides more semantics than the original relational schema. Therefore, an enhancement of the relational schema with more semantics is necessary. This can be achieved by querying the designer or administrator of the relational system or by using design information if available. Then the transformation will be done in a semi-automatic way. We will illustrate our ideas using the following example. Consider two databases, one relational database *Flight DB* containing information on passengers, flights, departures and their relationships, and an object-oriented database *Airport Personnel DB* containing information on ground personnel and flying personnel.

Flight DB:
Passenger (pid: Integer; name: String)
Departure (did: Integer; start: date; flight: Integer)
Pass_Dept (did: Integer; pid: Integer)

Airport Personnel DB:

```
Class Personnel with
    name, address: String;
    employeed_by: Company;
end;
Class GroundPersonnel
  isa Personnel with
    stationed_at: Airport;
end;
Class FlyingPersonnel
  isa Personnel with
    employeed_by: Airline;

Class Pilot isa
  FlyingPersonnel with
    can_fly: Plane;
end;
Class Plane with
    capacity: Integer;
    name: String;
end;
Class Airline isa Company
    . . .
end;
```

In this paper, we only focus on schema transformation and schema integration. This implies that only class definitions and mappings between classes and relational tables are created. Data, i.e. tuples, remain in the relational database. We do not consider problems like type mismatch, different metrics etc. These are for example tackled in other projects such as the Carnot [CHS 91] and Pegasus project [Ahm 91]. Furthermore, we restrict ourselves to classes whose instances are either completely managed by the object-oriented system or completely managed by the relational database.

3.1 Simple Transformation of the Relational Schema into Object-Oriented Class Definitions

The mapping of each relational table to a corresponding object-oriented class as done for example in the Pegasus project [Ahm 91] requires no additional information. Each table

$R(A_1: D_1; ..., A_n: D_n)$

is transformed into a class

Class R with
$A_1: D_1; ..., A_n: D_n$;
end;

This transformation is performed no matter what specific role the tables or attributes have. Table names become class names, attribute names become member variable names, and attribute domains become types of the member variables. We derive the following classes from our example tables:

```
Class Passenger with        Class Departure with        Class Pass_Dept with
    pid: Integer;               did: Integer;               did: Integer;
    name: String;               start: date;                pid: Integer;
end;                            flight: Integer;        end;
                            end;
```

All tuples of the tables become *virtual* instances of the respective classes of the object-oriented model. The object-oriented model does not provide more information than the relational. For querying the object-oriented schema, we have to reimplement the relational algebra. For example passengers and departures can be joined by the following nested loop:

```
for each pa in Passenger
 for each pa_de in Pass_Dept
  for each de in Departure
   if pa.pid = pa_de.pid and pa_de.did = de.did
   then return (pa.name, de.start, de.flight);
```

In this case, the object-oriented system is (ab-)used in a value oriented way. Unless an SQL-like or other declarative interface to the object-oriented system is provided, this is inefficient and not user friendly.

3.2 Transformation of Relational Schemas into Object-Oriented Class Definitions Using Meta Information

The object-oriented model provides better facilities to model the relationship of passengers and departures. It is not necessary to use value based references such as the attributes *pid* and *did* connecting passengers with departures but we may use the system inherent object identifiers and thus directly reference objects. This results in the following definition of *Pass_Dept*:

```
Class Pass_Dept with
    pass: Passenger;
    dept: Departure;
end;
```

The classes *Passenger* and *Departure* remain as before. The corresponding query is:

```
for each pd in Pass_Dept
  return (pd.pass.name, pd.dept.start, pd.dept.flight);
```

Although this query is very short, it is not intuitive in the sense that we have to use the class *Pass_Dept* which does not represent 'natural' objects. It would be more intuitive to ask the query from a passenger's point of view: "Give me all passengers, the start times and flight numbers of their departures." Using the set constructor we may omit the artificial class *Pass_Dept*:

```
Class Passenger with
    pid: Integer;
    name: String;
    dept_set: Set (Departure);
end;
```

```
Class Departure with
    did: Integer;
    start: date;
    flight: Integer;
    pass_set: Set (Passenger);
end;
```

Now, the corresponding query is:

for each pa in Passenger
{return pa.name;
for each de in pa.dept_set
return (de.start, de.flight)}

The last two solutions are better suited for the object-oriented model because the passenger-departure relationship is reflected directly and a more intuitive access from passengers to departures and vice versa using the set-oriented member variables is possible. Since these solutions contain more semantics than the original relational schema, any transformation process needs more input than the pure relational schema to produce such class definitions. To provide these additional semantics, is one of the goals of the model for meta information representation introduced in chapter 2. The meta information model does not only allow to specify arbitrary databases, tables and their attributes as instances. It also allows to specify whether they represent entities or relationships and additionally the type of relationships by using the member variables 'connecting_tables' and 'connecting_attributes' defined within the model. In our example, an instance of the member variable 'connecting_tables' classifies the table *Pass_Dept* as an m:n relationship joining tables *Passenger* and *Departure*. Furthermore, instances of member variable 'connecting_attributes' indicate that the join has to be carried out using the *pid* attributes of *Passenger* and *Pass_Dept* on the one hand and the *did* attributes of *Departure* and *Pass_Dept* on the other hand. This enables a schema transformer to replace join attributes and join tables by direct object references, e.g. by using the set constructor as pointed out in the above example.

As a consequence of the two tasks, i.e. instantiating the meta information model and creating object-oriented classes thereof, we work on a transformation algorithm consisting of two steps (see Figure 1). The first step is the semi-automatic enrichment of the relational model. It transforms the relational schema into instances of the meta model que-

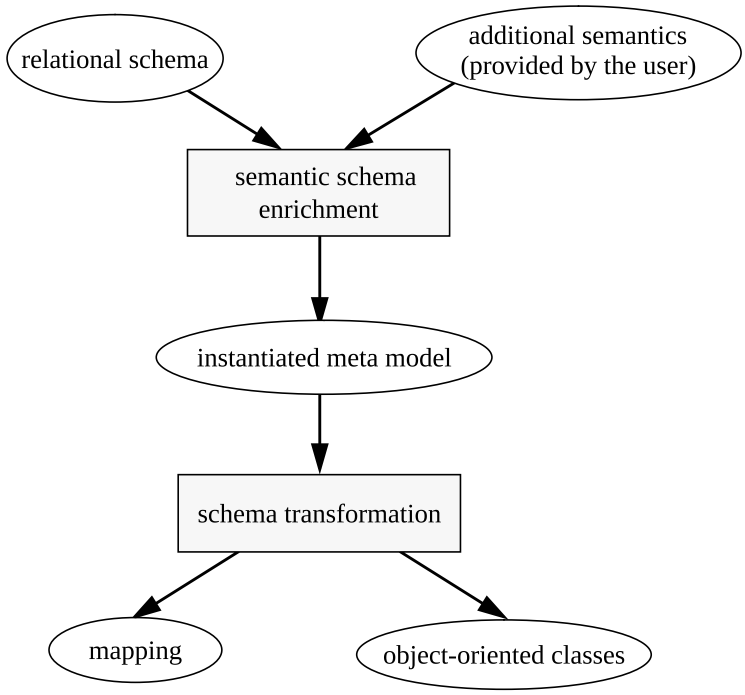

Figure 4: Two-step Transformation of a Relational Schema into Object-oriented Class Definitions

rying the administrator of the relational database for the necessary additional semantic knowledge such as

- tables representing relationships,
- the type of the relationship (1:1, 1:n, n:m),
- attributes or groups of attributes representing foreign keys.

The second step, the schema transformation, takes the instantiated meta model as input and produces the corresponding object-oriented classes and mappings. These mappings are necessary later on to support operations (methods) on the object-oriented classes. Recall that in the object-oriented system we only generate class definitions for the integrated tables, whereas the instances remain in the relational database. Thus access operations to instances of object-oriented classes have to be transformed into accesses to the corresponding relational tuples. Our approach to establish the mapping is by automatically linking each new class definition to its corresponding description in the meta model using a member variable ‘transformed_from’. This member variable is defined for every class and may be instantiated only once during class creation time. It is an attribute of the class definition rather than an attribute of each instance of that class. If we want to execute methods on instances of one or more classes, we can access the information provided by the meta model by following the ‘transformed_from’ links. This means, we can determine from which relational table the class is transformed, in which database we have to look for the instances and whether a given class attribute has to be translated to a join on the relational side.

Very often, before implementing a database, the domain of interest is formalized using an entity- relationship (ER) model. It can be observed that this model contains the semantic information we need for the first step of the transformation process. If there is a formalized and standardized semantic design model together with an also standardized mapping which entity and which relationship lead to which table, a nearly automatic schema enrichment is possible (see Figure 1). In prac-

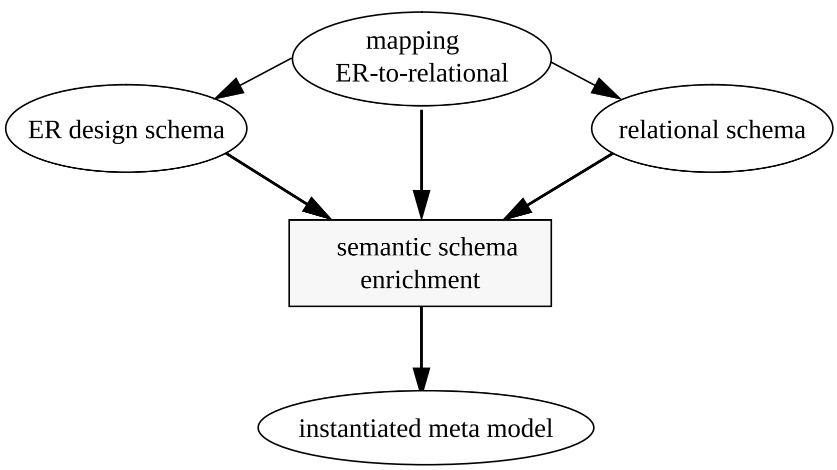

Figure 5: Semantic Schema Enrichment Improved by Information Obtained from the ER Design Schema

tice, there is no standardization for the mapping between the initial ER model and the resulting tables. Thus, user support will still be necessary to provide the information in a format suitable for the transformation process. But at least the user may be guided, for example by a graphical tool, to relate the ER design schema to the relational schema indicating the correspondent information in both schemas.

3.3 Inter-Database Relationships

For a good integration it is important that integrated classes of different databases can interact with already existing classes. In our example, we may for example want to establish e.g. a relationship between the originally object-oriented class *Pilot* and the class *Departure* integrated from the relational database. We have to deal with the following problem arising from the value based nature of the relational database and the object identifier based nature of the object-oriented system. If we add a member variable 'scheduled_for' of type *Departure* to the class *Pilot* then we must be able to assign to the 'scheduled_for' attribute of a concrete instance *p* of *Pilot*, a concrete instance *d* of *Departure*, i.e. we must be able to perform *p.scheduled_for := d*. This means, we have to solve the problem that a departure instance is uniquely identified by its key value *did* in the relational system but has no real object identifier. In general, on the object-oriented side we work with object identifiers which in addition are invisible to the user whereas on the relational side we work with keys built by groups of attributes.

Our proposal is to introduce an artificial class *Departure_Map* with member variables which are the key attributes of the *Departure* table. This class is instantiated for each tuple of table *Departure*, to which an instance of class *Pilot* is referring. The assignment statement *p.scheduled_for := d* is now transformed into:

```
Class Departure_Map with
    did: Integer;
end;
```

```
Departure_Map dm;
dm.create;
dm.did := d.did;
p.scheduled_for := dm;
```

This implies that we have to follow an extra link and, more seriously, that we have to maintain the instances of *Departure_Map*. Every time the relational database deletes a departure tuple, the objects of *Departure_Map* possibly reference a tuple that no longer exists. The maintenance task may be expressed by the following referential integrity constraint which is defined in the object-oriented system but also affects data in the relational database:

$$\forall\ x \in \text{Departure\_Map}\ \ \exists\ y \in \text{Departure} : (\text{x.did} = \text{y.did})$$

The usual strategy to satisfy such an integrity constraint is to delete all instances of *Departure_Map* referencing *y* before the departure tuple *y* itself may be deleted. However, in our case the relational database is an autonomous component and basically cannot be hindered from any deletion which afterwards the federated system has to react on.As a consequence, we must give up the policy:

"*Try an update (like deletion of Departure tuple y), check the integrity constraints, and if there is a violation reject that update*."

Instead, we have to adopt the new policy:

"*If the update of a component database violates an integrity constraint of the federated system, perform the necessary operations within the federated system in order to reach a consistent state of the overall system*."

A minimal requirement in this case is that the object-oriented system must be notified by the relational database of each update concerning inter-database relationships. But even this notification obligation interferes with the autonomy of the relational database and must be agreed upon within a 'contract' between the component and the federated system. If we integrate component databases which are not ready to sign such contracts, we must be able to handle a temporarily inconsistent state of the federated system.

4. Summary and Conclusions

A major problem in interoperating heterogeneous databases is necessary meta information on the databases participating in the federation. Unfortunately, such information is not readily available. Our concept of modeling the databases as object-oriented class hierarchy provides a consistent and effective way to store and access all necessary meta information. Part of the meta information supports a semantic enrichment of the databases allowing a (semi-)automatic schema and query translation as well as inter-database access. The meta information further supports a flexible specification of access rights and autonomy degrees and may also support the query optimization process. Finally, the meta information provides a simple but powerful support to the user in finding the desired databases. We believe that our approach to model meta information is elegant and effective because we use the object-oriented concept not only to store the information necessary to access and interoperate the heterogeneous databases, but also to enrich the semantic information available in their schemas. For the transformation of relational schemas into object-oriented class definitions, we use the meta information to provide an object-oriented class hierarchy with more semantics than the original relational schema. The missing semantic information can be determined by querying the designer or administrator of the relational database system or by using ER-design information if available. After completing the semantic enrichment and instantiating the meta model, the actual creation of the classes in the object-oriented model is done automatically. We are currently working on concepts to support a (semi-)automatic transformation of network and hierarchical schemas into object-oriented ones and on a (semi-)automatic integration of object-oriented schemas from different systems.

Further research is necessary to support not only the (semi-)automatic transformation of schemas but also to support an automatic translation of the data manipulation operations. This is important since the transformation of schemas has to be done only once, but the translation of operations has to be done every time a query is processed by the system. We envision a system with a uniform interface to query all databases participating in the federation. A preprocessor will use the meta information to detect accesses on virtual objects. The queries will be divided into parts corresponding to the different participating databases. The access of virtual objects will be translated into the data manipulation languages and sent to the underlying databases. A declarative query language for the object-oriented database system such as XSQL which is used in AXIS [Koj 91] or HOSQL which is used in Pegasus [Ahm 91] will be an essential part of our system for providing an easy, non-procedural access and query facility.

References

[Ahm 91] Ahmed R., De Smedt P., Du W., Knet W., Ketabchi M. A., Witwin W. A., Rafii A., Shan M.: *'The Pegasus Heterogeneous Multidatabase System'*, Proc. IEEE Computer, Vol. 24, No. 12, 1991, pp. 19-27.

[BLN 86] Batini C., Lenzerini M., Navathe S. B.: *'A Comparative Analysis of Methodologies for Database Schema Integration'*, ACM Computing Surveys, Vol. 18, No. 4, 1986, pp. 323-364.

[CHS 91] Collet C., Huhns M. N., Shen W.: *'Resource Integration Using a Large Knowledge Base in Carnot'*, IEEE Computer Vol. 24, No. 12, 1991, pp. 55-62.

[Cod 79] Codd E. F.: *'Extending the Database Relational Model to Capture More Meaning'*, ACM Trans. on Database Systems, Vol. 4, No. 4, 1979, pp. 397-434.

[CS 91] Castellanos M., Saltor F.: *'Semantic Enrichment of Database Schemas: An Object Oriented Approach'*, Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems, Kyoto, 1991, pp. 71-78.

[CT 91] Czejdo B., Taylor M.: *'Integration of Database Systems Using an Object-Oriented Approach'*, Proc. 1st Int.

Workshop on Interoperability in Multidatabase Systems, Kyoto, 1991, pp. 30-37.

[DA 83] Dumpala, Arora: *‘Schema translation using the Entity Relationship Approach’*, Entity Relationship Approach to Information Modeling and Analysis, Amsterdam, 1983.

[DA 87] Davis, Arora: *‘Converting a Relational Database Model into an Entity Relationship Model’,* Proc. 6th ER Conf., New York, 1987.

[EWH 85] Elmasri R., Weeldreyer J., Hevner A.: *‘The category concept: An extension to the entity-relationship model’,* Data & Knowledge Engineering, North-Holland, 1985, pp. 75-116.

[HK 88] Hull R., King R.: *‘Semantic Database Modeling: Survey, Applications, and Research Issues’,* ACM Computing Surveys, Vol. 19, No. 3, 1987, pp. 201-260.

[Ita 91] Itasca Systems Incorporated: *‘Technical Summary Release 2.0’*, Itasca System Incorporated, 1991.

[HLR 90] Holtkamp B., Lum V., Rowe N. C.: *‘DEMOM - A Description Bases Media Oject Data Model’,* Proc. Int. Computer Software and Applications Conference COMPSAC ‘90, Chicago, 1990.

[KDN 90] Kaul M., Drosten K., Neuhold E. J.: *‘ViewSystem: Integrating Heterogeneous Information Bases by Object-Oriented Views’*, Proc. IEEE Int. Conf. on Data Engineering, 1990.

[Kim 90] Kim W.: *‘Introduction to Object-Oriented Databases’*, Prentice Hall, 1990.

[Koj 91] Kojima I., Tanuma H., Sato Y., Ebihara I., Mano Y.: *‘Implementation of an Object-Oriented Query Language System with Remote Procedure Call Interface’,* Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems, Kyoto, 1991, pp. 79-86.

[LR 82] Landers T., Rosenberg R.: *‘An overview of Multibase’,* Distributed Databases, North-Holland, 1982.

[Mar 87] March S.: *'Entity-Relationship Approach'*, Proc. 6th ER Conf., New York, 1987.

[MM 90] Markowitz V., Makowsky J.: *'Identifying Extended Entity Relationship Object Structures in Relational Schemas'*, IEEE Tran. on Software Engineering, Vol. 16, No. 8, 1990.

[NA 87] Navathe S., Awong: *'Abstracting Relational and Hierarchical Data with a Semantic Data Model'*, Proc. 6th ER Conf., New York, 1987.

[Shi 81] Shipman D.W.: *'The functional data model and the data lanuage DAPLEX'*, ACM Trans. on Database Systems, Vol. 6, 1981, pp. 140-173.

[TYF 86] Teorey T. J., Yang D., Fry J. P.: *'A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model'*, ACM Computing Surveys, Vol. 18, No. 2, 1986, pp. 197-229.