Integration of Relational Databases in a Multidatabase System based on Schema Enrichment

Daniel A. Keim, Hans-Peter Kriegel, Andreas Miethsam
Institute of 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 and its use for the integration of heterogeneous databases with the goal of their interoperation. The meta information consists of all types 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. Additionally, we also include information to semantically enhance the schemas of the participating databases providing the basis for a (semi-)automatic schema transformation. We describe the semantic enrichment of a relational schema using additional information deduced from its underlying entity-relationship design schema. The enhanced relational schemas may be automatically transformed into corresponding schemas in the common data model which in our case is the object-oriented model. Queries using the created object-oriented schema may be automatically translated into equivalent SQL queries for the original relational schema.

Keywords: *interoperability of database systems, federated databases, multidatabase systems, schema enrichment, schema transformation, translation of operations, object-oriented database query language*

1. Introduction

The interoperation of heterogeneous database systems is necessary to provide convenient access to data distributed across different, already existing database systems. For many applications, interoperation based on logical data integration rather than physical data exchange is desirable. The need for a partial logical integration of databases is widely recognized but many difficulties arise when realizing it [KS 91] [BHP 92]. On the one hand the heterogeneous database schemas have to be transformed into the global common data model, and on the other hand the queries expressed in the global common data manipulation language have to be decomposed and translated into the local query languages. It is desirable that both processes, schema transformation as well as query translation, are highly automated. However, it is generally impossible to achieve a fully automated schema transformation since semantic information on the participating databases is missing. In many cases, it may be possible to deduce this information e.g. from existing entity-relationship design schemas, but in the other cases it must be user provided. In the process of query translation, user interaction is not feasible because query translation has to be done each time, a query is processed by the system. The possibility for queries to be automatically translated into the query languages of the participating databases is an important requirement for any global common query language. Another important requirement is the expressiveness of the global common query language since it should provide the possibility to express all queries which may be expressed in any of the participating databases. The global common query language itself depends on the common data model which must provide adequate concepts to model the semantic information 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] [CS 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. Research prototypes and commercially available object-oriented database systems 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 transformation because the same real world object may be modeled as entity in one schema but as a relationship in another schema. In the object-oriented data model, everything is modeled as an object and therefore, it is easier to integrate different schemas. An

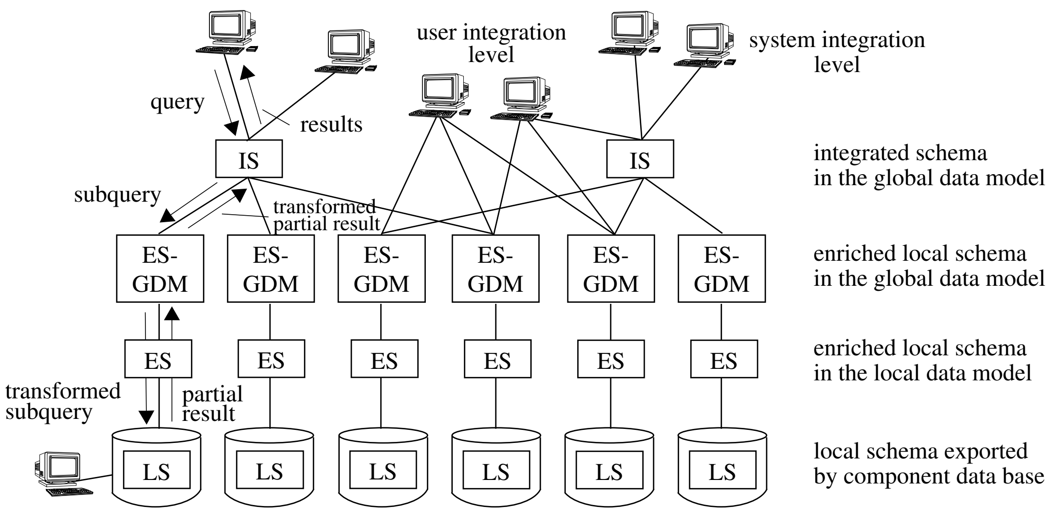

Figure 1: Schema Architecture

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 in the relational model.

In order to (semi-)automatically create more structured classes from relational schemas, in our approach, we use 'meta information' for semantic schema enrichment. In today's database systems, the so-called meta information is usually kept in some type of data dictionary. It is easy to store the meta information in the data dictionary as long as one is dealing with a system using a single data model. In a multidatabase environment, however, the meta information is structured differently in the various databases to be integrated. To adequately model meta information, we use an object-oriented class hierarchy for storing information on the participating databases. The meta information includes information necessary for an interoperation of the participating databases. It further provides a basis for a semantic enrichment of the schema information available in the participating databases. The additional semantic information is essential to support an automatic schema transformation from the original data model to the integrating common data model.

As already mentioned, it is important not only to support the enrichment of schema information and the automatic transformation of schemas, but also to support the automatic translation of 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 each time a query is processed by the system. To query the multidatabase system, we use a declarative query language called Structured Object Query Language (SOQL) allowing partial integration of heterogeneous databases and automatic translation of operations into the (possibly also high-level) query language of the participating databases. SOQL is similar to other object-oriented database query languages like OQL [ASL 89], XSQL [Koj 91] or HOSQL [Ahm 91] the latter being used in Pegasus.

Figure 1 shows the schema architecture of our system. It consists of five levels: the local schema (LS), the enriched local schema (ES), the enriched local schema in the global data model (ES-GDM), the integrated schema (IS) and the user level. The first level of our architecture is the local schema (LS) which is the exported part of the schema of a component database expressed in the local data model. Usually, the LS will be a view (subset) of the component database, but it may also be the complete schema without any modification. The LS may then be enriched with additional semantic information which may be user provided or deduced from an entity-relationship design schema. The result of this process is the enriched local schema (ES). The ES may then be (semi-)automatically translated into the global data model resulting in an enriched local schema in the global data model (ES-GDM). Several ES-GDM may be composed into an integrated schema (IS). The IS may be directly used by some users (system integration level) whereas others might prefer to access and integrate the ES-GDM on their own (user integration level), in which case the five level architecture shrinks to four levels. It further means that we support a tight coupling of the component databases but also allow the user to dynamically create new integrated schemas and to directly access the component databases, thereby also supporting a limited type of loose coupling. ES and ES-GDM are stored as part of the object-oriented (global) meta information already mentioned. The IS may also

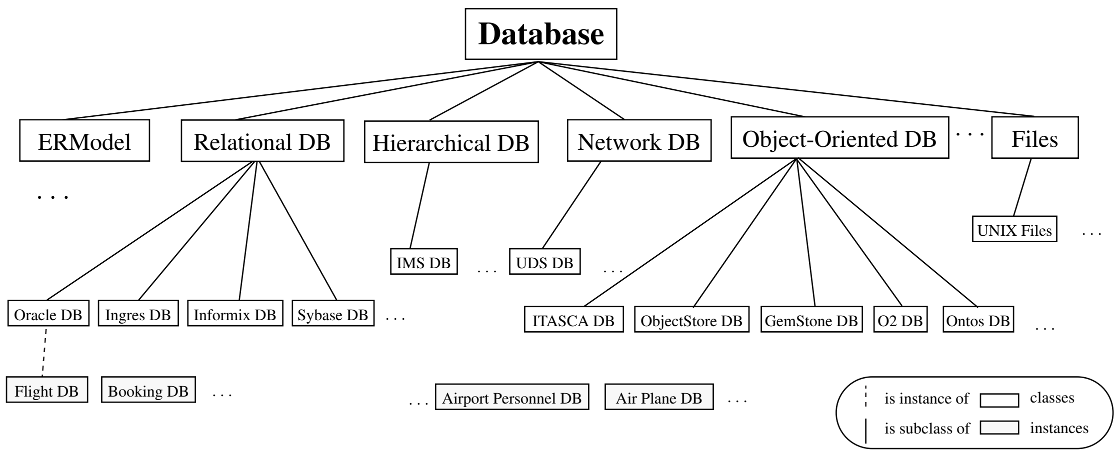

Figure 2: Object-Oriented Modeling of the Multidatabase

be part of the global meta information but usually they are stored locally (local meta information) because different applications and/or users may need different IS. Queries in SOQL using the IS are decomposed into SOQL-subqueries on the ES-GDM. Each of the subqueries is then translated into a query for the original schema (LS) in the corresponding data model. In the translation process, the additional semantic information stored as part of the global meta information is needed because the subqueries are directly translated from the ES-GDM to the LS. All subqueries are executed by the local database systems with the order of execution being defined by the multidatabase system. The (partial) results are transformed back into the global data model and assembled into the final result.

In the literature, several schema architectures have been proposed [Dev 82] [Lit 82] [Bla 87] [Tem 87] [RC 89]. Most of them are similar to the five level schema architecture described in [SL 90]. Our approach differs from this architecture mainly in two respects. First, in our architecture we do not need the export schema level because we consider the local schema (LS) as the exported subset of the component schema that is available to the multidatabase system. Second, we use a semantic schema enrichment for achieving a good integration. We, therefore, introduce an additional level with the enriched local schema (ES) which may be translated almost automatically into the global data model.

The rest of the paper is organized as follows: Section 2 introduces the object-oriented modeling of meta-information. This is the basis for access and interoperation of the participating databases and for the semantic enrichment of their schemas. Section 3 elaborates on the (semi-)automatic transformation of relational schemas into object-oriented ones using meta information deduced from the underlying entity-relationship schema. In section 4, we show that the user has full SQL-like access to the underlying relational databases using SOQL queries which may be automatically translated into equivalent SQL queries. Section 5 summarizes our approach and points out some problems related to an automatic translation of data manipulation operations in a multidatabase environment.

2. Modeling the Databases Participating in the Multidatabase

To adequately model meta information, we use an object-oriented class hierarchy for the participating databases, which we presented in our paper [KKM 92]. This section gives a brief overview and some additional details of the object-oriented system described there. The object-oriented system uniformly manages the following tasks of a multidatabase system:

- It models all databases participating in the multidatabase. Instances of the meta model are objects describing databases, their data models and system properties. The meta model and its instances represent the knowledge base of the multidatabase management system.
- It allows to define and work with the integrated and/or enriched local schema in the global data model consisting of class definitions. Each class definition has corresponding instances in the meta model which it refers to by an object reference called 'transformed_from'. Instances of these classes are "real data" which either can be found in the object-oriented system or in the participating component systems.

The knowledge base part, i.e. the meta model which will be presented in the rest of this section, serves for

- integrating (parts of) the data dictionaries of the participating databases (including the data dictionary of the multidatabase system itself),
- enriching the schemas of the participating databases,
- maintaining transformation and access information,
- dynamically changing parameters of a database during the time the database participates in a federation,
- integrating new database systems and data models not yet known to the system.

2.1 The Class Hierarchy for Modeling the Participating Databases

The mentioned class hierarchy implements on the class level a model for arbitrary data dictionaries, so to speak, a data dictionary for data dictionaries. At the instance level, it contains the contents of the data dictionaries of the participating databases with additional information necessary for managing the multidatabase system. In other words, at the instance level it contains the enriched schema (ES) of the local data model (see Figure 1).

The information on common properties of these databases as well as differences between them is represented by class attributes and functions or member variables and methods as they would be called in object-oriented database systems based on C++. The class hierarchy is presented in Figure 2, where boxes depict class definitions including member variables leading to other class definitions not shown in this figure. In the hierarchy, the class 'Database' has subclasses 'Relational DB', 'Network DB', 'Hierarchical DB', 'Object-oriented DB' and 'Files' for simple file systems, each of them representing a specific data model. These classes further branch into subclasses according to the concrete (commercial) systems available. 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. Additionally, we include the class 'ERModel' in our model, since many databases have been designed using an entity-relationship-diagram. The knowledge of both, the entity-relationship-diagram and the concrete database schema allow a higher degree of automation of the schema transformation process, as will be pointed out in the next section.

For each database type the meta model also contains classes which model the building elements of this type. In the relational case classes such as 'Table', 'Attribute' provide the frame to store all necessary information on the respective database. Figure 3 shows some important classes in detail and gives an impression how meta-information is represented in our system. An outcome of this model is that information is treated in a unique way where it is possible. For example, consider 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, Sybase, 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). Our system, however, should be able to operate on such result tables as well. Using the subclass mechanism is a good approach since the class 'Table' inherits all properties specific to the corresponding data model (i.e. the relational model) but has no information related to a concrete database system.

2.2 Semantic Enrichment of Local Data Dictionaries

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. Additional classes, special member variables and subtyping are used to enhance the semantic information available in the data model or concrete system. An example for enhancing the relational model (see Figure 3) with information on tables implementing m:n, 1:n or 1:1 relationships are:

- the member variables
  - 'connecting_tables' defined in class 'Relational DB', leading to all tables establishing relationships between tables representing entities,
  - 'connecting_attributes', 'fct_relationships' defined in class 'Relational DB', leading to information, which tables can be joined using which attributes,
  - 'connected_tables' defined in class 'ConTable', leading to all tables which are connected by the respective instance of 'ConTable',
- the subtype hierarchy 'SimpleConTable' isa 'ConTable' isa 'Table', explicitly distinguishing two classes of relationship tables and entity tables. Instances of the class 'SimpleConTable' indicate tables which should not be transformed into separate classes but may be replaced by member variables which directly reference the related class,
- the additional classes 'FctRelationship', 'TableRelationship', providing details for joining two tables, where instances of 'FctRelationship' denote 1:n relationships.

This semantic information on relationships between tables and on corresponding attributes is not modeled explicitly in the relational model. However, it is necessary for an adequate schema transformation as well as for querying the database based on the transformed schema with enriched information. The additional semantic information is necessary to automate schema translation and integration of the database schemas using different data models and database systems. The (semi-)automatic schema translation process from the relational to the object-oriented model will be described in the next section. Vice versa, to translate operations on the transformed schema we also need this information to correctly transform back object references into operations of the local databases. Furthermore, it enables the interoperation of the databases and supports the access of several databases using one query interface.

```
Class Database with
   name: String;
   description: Text;
   db_size: Integer;          # in MByte
   owner: Owner_Spec;
   location: Network_Address;
   access_rights: Access_Spec;
   degree_of_autonomy: Autonomy_Spec;
end;
Class Relational DB isa Database with
   tables: Set(Table);
   views: Set(View);
   connecting_tables: Set(ConTable);
   connecting_attributes: Set(TableRelationship);
   fct_relationships: Set(FctRelationship)
end;
. . .
Class Table with
   name: String;
   description: Text;
   attributes: Set(Attribute);
   key: Set(Attribute);
   number_of_records: Integer;
constraints:
   this.key ⊆ this.attributes;
end;
Class Attribute with
   name: String;
   domain: Domain;
   description: Text;
   defined_in: Table;
end;
Class TableRelationship with
   tab1, tab2: Table;
   joinattr1: List(Attribute);
   joinattr2: List(Attribute);
constraints:
   this.joinattr1.count = this joinattr2.count;
   this.joinattr1.domain = this joinattr2.domain;
   this.joinattr1.listtoSet ⊆ this.tab1.attributes;
   this.joinattr2.listtoSet ⊆ this.tab2.attributes;
end;
Class FctRelationship isa TableRelationship with
constraints:
   tab1 renamed_to from_tab;
   tab2 renamed_to to_tab;
   ∀ tuple t1 ∈ this.from_tab ∃1 tuple t2 ∈ this.to_tab:  t1 ⋈ t2 ;
end;
Class ConTable isa Table with
   connected_tables: Set(Table);
   connecting_attributes: Set(TableRelationship);
end;
Class SimpleConTable isa ConTable with
constraints:
   this.connected_tables.count = 2;
   this.attributes = this.connecting_attributes.join-
            attr1.unnesting; (*all attributes are join attributes*)
end;
Class ERModel with
   name: String;
   description: Text;
   specification_for: Database;
   entities: Set(Entity);
   relationships: Set(Relationship);
end;
Class Relationship with
   name: String;
   description: Text;
   connected_entities: Set(Entity);
   attributes: Set(ERAttribute);
end;
Class Entity with
   attributes: Set(ERAttribute);
end;
. . .
```

Figure 3: Object-Oriented Modeling of the Multidatabase

3. Enrichment and Transformation of Relational Schemas

In this section, we investigate how the schema of a relational database can be transformed into class definitions in an object-oriented model. Usually, a good object-oriented schema contains more semantics than the relational schema for the same application domain. If an automatic transformation process is aimed to produce adequate, well-structured object-oriented class definitions, more input than the pure relational schema is needed. The modeling of the necessary additional semantics is one of the goals of the meta information representation introduced in section 2. The meta information model does not only allow to specify arbitrary databases, tables and their attributes as instances. Additionally, it also allows to specify whether they represent entities or relationships and furthermore the type of relationship by using the member variables ‘connecting_tables’ and ‘connecting_attributes’ defined within the model.

For illustrating the schema enrichment and transformation process, we will use the following example. Consider a relational database *Flight DB* containing information on passengers, departures, planes, planetypes and their relationships.

Flight DB:

Passenger (pid: Integer; name: String; address: String)
Departure (did: Integer; start: date; flight: Integer;
airline_id: Integer; plane_id: Integer)

Pass_Dept (did: Integer; pid: Integer)
Plane (serial_nr: Integer; yr_built: Date;
manufacturer: String; model: Integer)
Planetype (manufacturer: String; model: Integer;
capacity: Integer; range: Integer)

For this database, an instance of the member variable ‘connecting_tables’ would classify the table *Pass_Dept* as an m:n relationship joining tables *Passenger* and *Departure*. Furthermore, an instance of the member variable ‘connecting_attributes’ indicates 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 additional semantic information is crucial for the schema transformation process in order to create good object-oriented schemas by replacing connecting attributes and connecting tables by direct object references.

3.1 Schema Enrichment

As already indicated, the semantic enrichment of a relational schema corresponds to the instantiation of the meta model (see section 2) with information on the relational schema and 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.

This information can be obtained by querying the designer or administrator of the relational system. In the case of relational databases, very often the domain of interest is formalized using an entity-relationship (ER) model or, at least, the ER schema may be (partially) deduced from the relational schema [MM 90]. This model contains the semantic information needed for our schema enrichment. 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. In the following, we formally describe the schema enrichment if a standard ER schema [Che 76] is available. Let us first consider the three steps in the process of translating an ER schema into a relational schema:

1. for each entity $E(X_1, \dots X_m, A_1, \dots, A_n)$ with $(m+n > 1)$
$\Longrightarrow$ Relation $E(X_1, \dots X_m, A_1, \dots, A_n)$ is created.
2. for each functional relationship $R: E \rightarrow F$ with $y_1, \dots, y_f$ key of F and $E(A_1, \dots, A_q)$
$\Longrightarrow$ Relation E is extended to $E\ (A_1, \dots, A_q, y_1, \dots, y_f)$ and no relation for R is created.
3. for all other relationships $R(E_1, \dots, E_p, A_1, \dots, A_q)$ with $a_{i1}, \dots, a_{in_i}$ key of E_i
$\Longrightarrow$ Relation R $(a_{11}, \dots, a_{1n_1}, \dots, a_{p1}, \dots, a_{pn_p}, A_1, \dots, A_q)$ is created.

When translating entities into relations (step 1) no semantic information is lost. In the second step, however, the information that there exists a functional relationship between E and F gets lost during the transformation process. Therefore we add this information to our meta model by instantiating the class *FctRelationship* with [from_tab=E; to_tab=F; joinattr1=$(y_1, \dots, y_f)$; joinattr2=$(y_1, \dots, y_f)$] and by adding this instance to the member variable *fct_relationships* of the corresponding database. We assume that we have already instantiated E and F as instances of class *Table,* $y_1, \dots, y_f$ as instances of class *Attribute* and so on. For simplification, we further assume that after translating the ER to the relational schema no attributes or tables have been added, deleted or renamed, nor that tables have been omitted or united. In the third step, even more information is lost because the p-ary relationship between $E_1, \dots, E_p$ is only represented indirectly in the relational schema. Again, we use our meta model to capture the additional semantics. As before, we create instances of the class *TableRelationship* with J_i = [tab1=R; tab2=E_i; joinattr1=$a_{i1}, \dots, a_{in_i}$; joinattr2=$a_{i1}, \dots, a_{in_i}$] for i=1 ... p and add them to the member variable *connecting_attributes* of the corresponding database. Furthermore, we instantiate the class *ConTable* with [connected_tables=Set($E_1, \dots, E_p$); connected_attributes=Set($J_1, \dots, J_p$)] and add this instance to the member variable *connecting_tables*. If the relationship connects only two tables (p=2) and if it has no additional attributes (q=0) we use the class *SimpleConTable*, which is a subclass of *ConTable*. *SimpleConTable* is important for the schema transformation process since simple connecting tables may be omitted completely in the object-oriented schema.

In most cases, there is neither a standard ER-model nor a standardization for the mapping between the initial ER model and the resulting tables. Thus, user support will be necessary to instantiate the meta model, but in any case, part of the additional semantic information can be automatically deduced and the user may be guided in the process of relating the ER design schema to the relational schema.

3.2 Schema Transformation

Having the additional semantic information instantiated in the meta model, the schema transformation process may be performed automatically. The schema transformation uses the instantiated meta model as input and produces the corresponding object-oriented classes and mappings. The first step of the schema transformation is a simple transformation of all relations into classes. For a database *DB*, each table $T \in$ *DB.table*

T(A_1: D_1; ..., A_n: D_n) is transformed into

Class T with
A_1: D_1; ..., A_n: D_n ;
end;

Together with the class definition, an access method with the same name is created for each member variable. Using this simple transformation, objects are only referenced by value. In the object-oriented model, however, relationships may be represented directly. Using direct object references or set-oriented member variables, a more intuitive access is possible.

Therefore, in the second step, we introduce new attributes for all functional relationships. For each $FR \in$ *DB.fct_relationships*, the classes *E* and *F* created from *FR.from_tab* and *FR.to_tab* are extended:

Class E with A_1: D_1; ..., A_n: D_n ; *end;*	to	*Class E with* A_1: D_1; ..., A_n: D_n ; *"F": F;* *end;*
Class F with A_1: D_1; ..., A_n: D_n ; *end;*	to	*Class F with* A_1: D_1; ..., A_n: D_n ; *"E_set": Set(E);* *end;*

"F" indicates that the default name of the additional member variable is the name of the corresponding class F and for set variables the class name is concatenated with "_set" which is indicated by "E_set".

In the third step of the transformation process, we consider arbitrary relationships. For each $CA \in$ *DB.connecting_attributes,* the classes *R* and E_i (created from *CA.tab1* and *CA.tab2*) are extended using a member variable directly referencing the other class. The class *R* (created from *CA.tab1*) corresponding to the relationship table in the ER schema is extended by a member variable directly referencing class E_i and the class E_i (created from *CA.tab2*) is extended by a member variable referencing a set of objects of class R. The formal description of the class extensions is similar to the second step when replacing *E* by *R* and *F* by E_i.

The last step of the schema transformation process is the transformation of simple connecting tables. For each $SCT \in$ *DB.SimpleConTable*, the classes E_1 and E_2 (created from *SCT.connected_tables*) are changed to have direct references to each other instead of referencing the class R as introduced in the last step. Formally, the classes E_1 and E_2 are changed as follows:

Class E_1 *with* A_1: D_1; ..., A_n: D_n ; *"R_set": Set(R);* *end;*	to	*Class* E_1 *with* A_1: D_1; ..., A_n: D_n ; *"*E_2*_set": Set(*E_2*);* *end;*
Class E_2 *with* A_1: D_1; ..., A_n: D_n ; *"R_set": Set(R);* *end;*	to	*Class* E_2 *with* A_1: D_1; ..., A_n: D_n ; *"*E_1*_set": Set(*E_1*);* *end;*

Having transformed E_1 and E_2, the class R is no longer needed and can be omitted.

Let us now consider the schema transformation for our example database. There, we have a functional relationship between Plane and Planetype and a simple connecting table Pass_Dept connecting passenger and departure. If we have the necessary semantic information instantiated in the meta model, we can use the schema transformation algorithm to produce the following class definitions representing an equivalent object-oriented schema:

Class Passenger with
pid: Integer;
name: String;
address: String;
departure_set: Set (Departure);
end;

Class Departure with
did: Integer;
start: date;
flight: Integer;
airline: Airline;
plane: Plane;
passenger_set: Set (Passenger);
end;

Class Plane with
serial_nr: Integer;
yr_built: Date;
model: Integer;
planetype: Planetype;
end;

Class Planetype with
manufacturer: String;
model: Integer;
capacity: Integer;
range: Integer;
plane_set: Set (Plane);
end;

Our schema transformation may not provide a perfect object-oriented schema. However, it creates a schema which allows direct access to the objects as it is usual in an object-oriented system. We found that SOQL queries using the created classes are often shorter and more intuitive than the corresponding SQL query using the original tables. Complex joins are replaced by simple accesses to member variables providing more natural access paths.

At this point, let us emphasize that we only generate class definitions in the multidatabase system, whereas the instances remain in the relational database. Thus, access operations to objects of object-oriented classes with underlying relational data bases have to be translated into accesses to the corresponding relational tuples. An advantage of our schema enrichment and transformation is that this translation can be done automatically which will be described shortly in section 4. For the automatic translation of operations, a mapping is needed between created class definitions and meta information used to create them. Our approach to establish the mapping is to link automatically 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. More exactly, we can determine from which relational table the class is transformed, in which database we have to look for the instances and whether accesses to class attributes have to be translated to joins on the relational side.

4. Translation of SOQL- into SQL-Queries

In this section, we want to illustrate the translation of SOQL-queries using the created object-oriented schema into SQL-queries using the original relational schemas.

First, we have to introduce our multidatabase query language SOQL. SOQL is defined for the global data model and has been designed to query the integrated schemas or single ES-GDMs. SOQL is similar to other declarative query languages for object-oriented database systems. As in most other object-oriented database query languages, in SOQL a set of basic object classes (Boolean, String, Numbers, Integer, Real and the generic classes Set and List) is defined. The classes are provided by the system together with a set of basic methods. Methods are applied to an instance variable of an object using dot-notation, e.g. adding 5 to an integer object with identifier int_ob is done by 'int_ ob.+(5)'. For a more convenient use of methods, the standard infix notation is allowed for the predefined methods of the basic classes. Additional methods for any class may be introduced by the user. These methods may be used like any of the system provided methods. However, when using these methods, there is no automatic translation to an SQL query in the underlying relational database. The system will issue SQL queries to access the data needed to execute the methods in the object-oriented system. As already shown in the example class definitions (see Figure 3), we assume that our object definition language allows to state many sorted first order integrity constraints. Due to space limitations, in this paper we do not elaborate on the capabilities of SOQL.

In the following, we will give two examples for queries in SOQL. For the example queries, we use the transformed example database as presented in section 3. A simple query selecting all flight numbers with a list of the corresponding passenger names for the airline "Lufthansa" on the 07/12/92 is expressed as follows:

Example 1:

```
select D.flight D.passenger_set.name
for each Departure D
where D.start = "07/12/92" and
      D.airline.name = "Lufthansa"
```

In the relational system, even in this simple case four tables need to be joined in order to execute the query. The second example is a query to determine the seat utilization of all "Lufthansa" flights. The following query produces the desired results:

Example 2:

```
select (D.plane.planetype.capacity - D.passenger_set.count)
for each Departure D
where D.airline.name = "Lufthansa"
```

In this query, we repeatedly use the dot-notation in a row. A corresponding SQL-query is much more complicated because joins between all six tables involved in the query are required.

SOQL queries have to be transformed into queries in the local data model, depending on which component databases have to be accessed. Let us recall that only schemas are integrated, the instances, however, remain in the local databases. Since an SOQL query may involve classes which represent data of different databases, the query must first be decomposed into homogeneous queries. In a homogeneous query, all parts of the query must belong to the same database and member variables or methods added to a class after its creation by the schema transformation process may not be applied to instances of the class.

It is possible to automatically translate homogeneous SOQL queries into queries defined in the local data model using the semantic and transformation information provided by the meta model. To illustrate the translation process, in the following, we will give an example. In the translation of query Example 1, successively class variables D.passenger_set and D.airline are replaced introducing the join conditions join(D, nv2) and join(D, nv1). Then the relational join expressions are created from the join conditions (adding Pass_Dept in this step) and the 'group by' clause is appended reflecting the set structure of the SOQL answer type.

Translation of Example 1:

```
select D.flight nv1.name
from Departure D, Passenger nv1, Airline nv2, Pass_Dept pd
where D.airline_id = nv2.aid and D.did = pd.did
      and pd.pid = nv1.pid and D.start = "07/12/92"
      and nv2.name = "Lufthansa"
group by D.flight
```

5. Summary and Conclusions

A major problem in interoperating heterogeneous databases is the necessary meta information on the databases participating in the multidatabase. Unfortunately, such information is not readily available. Our concept of modeling the databases as an 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 efficient 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 information deduced from an ER-design schema to provide an object-oriented class hierarchy with more semantics than the original relational schema. As described, the semantic enrichment and instantiation of the meta model as well as the actual creation of the classes in the object-oriented model may be performed automatically. For a convenient access to the data in the object-oriented multidatabase system, we use SOQL, a declarative query language for objects. SOQL provides a uniform and convenient query interface to all participating databases which, in addition, is easily extendable. For the purpose of interoperation, it is further important that SOQL has more expressive power than SQL and that SOQL queries which have corresponding SQL queries can automatically be translated into equivalent SQL queries.

We are currently working on concepts to support a (semi-) automatic enrichment and transformation of network and hierarchical schemas into object-oriented class definitions in order to integrate also these types of databases in the multidatabase. It is necessary to support not only the schema integration and management of all types of inter-database relationships, but also to automatically partition and translate SOQL queries on an integrated schema into the data manipulation operations of the underlying databases. In the process of query translation, the meta information will be crucial for finding the virtual object classes and the information necessary to access the underlying databases.

References

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

[ASL 89] Alashqur A. M., Su S. Y., Lam H.: *'OQL: A Query Language for Manipulating Object-Oriented Databases',* Proc. 5th Int. Conf. on Very Large Data Bases, Amsterdam, 1989, pp. 433-422.

[BHP 92] Bright M. W., Hurson A. R., Pakzad S. H.: *'A Taxonomy and Current Issues in Multidatabase Systems',* Proc. IEEE Computer, 1992, pp. 50-60.

[Bla 87] Blakey M.: *'Basis of a Partially Informed Distributed Database',* Proc. 13th Int. Conf. on Very Large Data Bases, Brighton, UK., 1987, pp. 381-388.

[Che 76] Chen P. P.-S.: *'The Entity-Relationship Model - Toward a Unified View of Data',* Proc. ACM Trans. on Database Systems, Vol. 1, No. 1, 1976.

[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, Japan, 1991, pp. 30-37.

[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, Japan, 1991, pp. 71-78.

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

[Dev 82] Devor C., Elmasri R., Larson J., Rahimi S., Richardson J.: *'Five-schema Architecture Extends DBMS to Distributed Applications',* Electron. Des., No. 18, 1982, pp. 27-32.

[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.

[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 6th Int. Conf. on Data Engineering, Los Angeles, CA, 1990, pp. 2-10.

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

[KKM 92] Keim D. A., Kriegel H.-P., Miethsam A.: *'Object-Oriented Modeling of Meta Information for Semantic Schema Enrichment and (Semi-) Automatic Schema Transformation',* Technical Report, Institute of Computer Science, University of Munich.

[KS 91] Kim W., Seo J.: *'Classifying Schematic and Data Heterogeneity in Multidatabase Systems',* Proc. IEEE Computer, Vol. 24, No. 12, 1991, pp. 12-18.

[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, Japan, 1991, pp. 79-86.

[Lit 82] Litwin W., Boudenant J., Esculier C., Ferrier A., Glorieux A., La Chimia J., Kabbaj K., Moulinoux C., Rolin P., Stangret C.: *'SIRIUS Systems for Distributed Data Management',* Distributed Data Bases, North-Holland.

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

[MM 90] Markowitz V., Makowsky J.: *'Identifying Extended Entity Relationship Object Structures in Relational Schemas',* IEEE Trans. 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.

[RC 89] Ram S., Chastain C.: *'Architecture of Distributed Data Base Systems',* Journal System Software, Vol. 10, No. 2, 1989, pp. 77-95.

[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.

[SL 90] Sheth A. P., Larson J. A.: *'Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases',* ACM Computing Surveys, Vol. 22, No. 3, 1990.

[Tem 87] Templeton M., Brill D., Chen A., Dao S., Lund E., McGregor R., Ward P.: *'Mermaid: A Front-End to Distributed Heterogeneous Databases',* Proc. IEEE Vol. 75, No. 5, 1987, pp. 695-708.

[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.