QUERY DECOMPOSITION AND PROCESSING IN MULTIDATABASE SYSTEMS

Sena Nural **Pinar Koksal** **Fatma Ozcan** **Asuman Dogac**

Software Research and Development Center of TUBITAK
Dept. of Computer Engineering
Middle East Technical University (METU)
06531 Ankara Turkiye
email: asuman@srdc.metu.edu.tr

ABSTRACT

A multidatabase system allows its users to simultaneously access heterogeneous, and autonomous databases using an integrated schema and a single global query language. An important problem in multidatabase systems is processing of the global queries. In this paper, we describe a global query processing scheme as it is implemented in a multidatabase environment, namely MIND. Since multidatabase query processing is very much dependent on the way schema integration is realized, the underlying MIND schema integration is also described. The details of both query decomposition process and the necessary post-processing to combine the partial results coming from local databases, are provided.

1 INTRODUCTION

Various types of database systems are currently in use today. These information resources could be immediately made available for many users through existing computer systems. However, since these systems often use different data models and different query languages, users of one system can not easily access the data stored in other systems. Thus, there is a growing need for tools to maximize the reusability and interoperability of arbitrary computing services while keeping the autonomy of pre-existing databases in a federated approach.

One way of achieving interoperability among heterogeneous and autonomous database management systems (DBMS) is to develop a multidatabase system which supports a single common data model and a single query language on top of different types of existing systems. The global schema of a multidatabase system is the result of the integration of schemas exported from the underlying databases (i.e. local databases). A global query language is made available to the users of the multidatabase system to specify the queries against the global schema which are called global queries.

Three steps are necessary to process global queries (Meng, W. and Yu, C., 1995): First, a global query is decomposed into a number of subqueries such that the data needed by each subquery are available from one local database. After the decomposition, each subquery is translated to a query or queries of the corresponding local database system and sent there for execution. Finally, the results returned from local DBMSs are combined into the answer.

An important point to note over here is the following: Multidatabase query processing is very much dependent on the way schema integration is realized. In other words, schema integration constitutes the base of multidatabase query processing.

In this paper, we describe schema integration process and query processing as it is implemented in a multidatabase system, namely MIND (Kilic, E. et al., 1995; Dogac, A. et al., 1995b; Dogac, A. et al., 1996). We give the details of query decomposition process and the post-processing operations that are required to combine the partial results coming from local databases. Query optimizer component of MIND is described in (Evrendilek, 1995; Evrendilek, C. et al., 1995b; Evrendilek, C. et al., 1995a; Ozcan, 1996).

The paper is organized as follows : Section 2 summarizes the related work on query processing and schema integration. Section 3 describes the implementation framework, namely MIND architecture. Section 4 addresses the schema integration in MIND. In Section 5, we give the detailed description of query decomposition. Combination of the partial results to form the final answer is explained in Section 6. Finally, conclusions and the future work are given in Section 7.

2 RELATED WORK

Since schema integration is directly related with multidatabase query processing, we first describe the related work on schema integration.

A general classification of schematic conflicts that may arise when integrating relational and object-oriented databases is provided in (Kim, 1995). In this paper, a comprehensive classification of conflict resolution techniques are also given. It is stated that the extensions to traditional DDLs and DMLs are necessary to define a multidatabase schema that will resolve various schematic conflicts in component databases (CDB). A multidatabase language SQL/M which is an extension of SQL/X is used. In SQL/M, the user can integrate one or more CDB entities into a virtual class.

(Busse R. et al., 1994) introduces the notion of virtual classes into the object database standard ODMG-93. These virtual classes can be used to establish a federation database which is capable of importing and exporting data from and to other databases, to integrate information from different bases, and to tailor the overall schema for specific applications.

Query modification when generalization is used for schema integration is discussed in (Dayal, 1985). Generalization is modeled algebraically as a sequence of outerjoins and aggregation operations. It is stated that conventional database systems use a variety of tactics one of which is to perform selections and projections at individual sites. However, these tactics are of limited applicability in a multidatabase system where queries can include outerjoins and aggregates. To make local selections possible, the semiouterjoin technique is proposed. Semiouterjoin operation produces two results, semijoin and its complement. Using the semiouterjoin operation, relations are partitioned into two subsets, private portion and overlap portion. Then, it becomes possible to perform local selections on the private portion.

In (Scheurmann, P. and Chong, E., 1994), role-based query processing in multidatabase systems are considered and methods are proposed. In this approach, the answer to a query is given as a set of sets representing distinct intersections between the relations corresponding to the various roles. This set is called as a role-set. MSQL is extended to make it possible to express role sets and quantifiers applied to them. This strategy allows for local selection of all queries with/without aggregation.

Another query processing strategy is explained in (Finance, B. et al., 1995) within the scope of an ESPRIT project IRO-DB. The main idea in this strategy is the use of a home database which contains some useful information for query processing. Objects that are referenced in queries are replicated partially or totally on the home OODBMS that improves the efficiency of a running application by reducing the remote accesses and object transfers. After a delay, the system behaves like a centralized system where all imported classes are fully instantiated in the Home OODBMS. In this paper, optimization rules are proposed for different query types and instantiation levels trying to minimize the remote accesses to local DBMSs.

3 AN ENVIRONMENT FOR MULTIDATABASE QUERY PROCESSING

The query processing approach presented in this paper is implemented in the MIND (METU Interoperable DBMS) project (Dogac, A. et al., 1995b; Dogac, A. et al., 1995c; Dogac, A. et al., 1996; Kilic, E. et al., 1995). MIND architecture is based on OMG's[1] distributed object management architecture (Soley, 1992). The infrastructure of the system is build on a CORBA implementation, namely DEC's ObjectBroker[2] (DEC, 1994). An interface of a generic Database Object through CORBA IDL has been defined and multiple implementations of this interface for Sybase[3], Oracle7[4], Adabas D[5] and MOOD (METU Object-Oriented DBMS) (Dogac, 1994; Dogac, A. et al., 1995a) have been developed. Thus, what clients of this level see are homogeneous DBMS objects accessible through a common interface. The main architecture of MIND is given in Figure 1. The basic components of the system are a Global Database Agent (GDA) class and a Local Database Agent (LDA) class. When a client wants to interact with MIND, a GDA object (GDAO) is created by sending a create message to the Object Factory. ORB provides the location and implementation transparency for GDAOs. The location of GDAOs are dynamically determined by the ORB using the information provided by the ORB administrator. The default site for the GDAO is usually the local host to prevent communication costs. A GDAO contains objects of two classes, one responsible from global transaction management, namely Global Transaction Manager (GTM) class and the other from global query processing, namely Global Query Manager (GQM) class. A GTM object (GTMO) and a GQM object (GQMO) are created for each session of each client. GQMO obtains the global schema information necessary for the decomposition of the query from the SchemaInformationManager object (SIMO). After decomposing, GQMO sends the subqueries to GTMO. GTMO is responsible from the correct execution of global transactions. For this purpose, it modifies the global subtransactions when necessary and controls the submission of global subtransations to the LDAOs. LDAOs control submission of operations to the LDBMSs and communicate with GTMO and GQMO to achieve atomic commitment of global transactions. Note that the LDAOs execute the global subqueries in parallel. In this architecture there is another class, called the Query Processor class responsible from processing the partial results returned by the LDAOs. As soon as two partial results that can be processed together appear at the LDAOs, a Query Processor Object(QPO) is created to process them. There could be as many QPOs running in parallel as needed to process the partial results.

[1] OMG is a registered trademark, and CORBA, ORB, OMG IDL, Object Request Broker are trademarks of OMG.

[2] ObjectBroker is a registered trademark of DEC.

[3] Sybase is a trademark of Sybase Corp.

[4] Oracle7 is a trademark of Oracle Corp.

[5] Adabas D is a trademark of Software AG Corp.

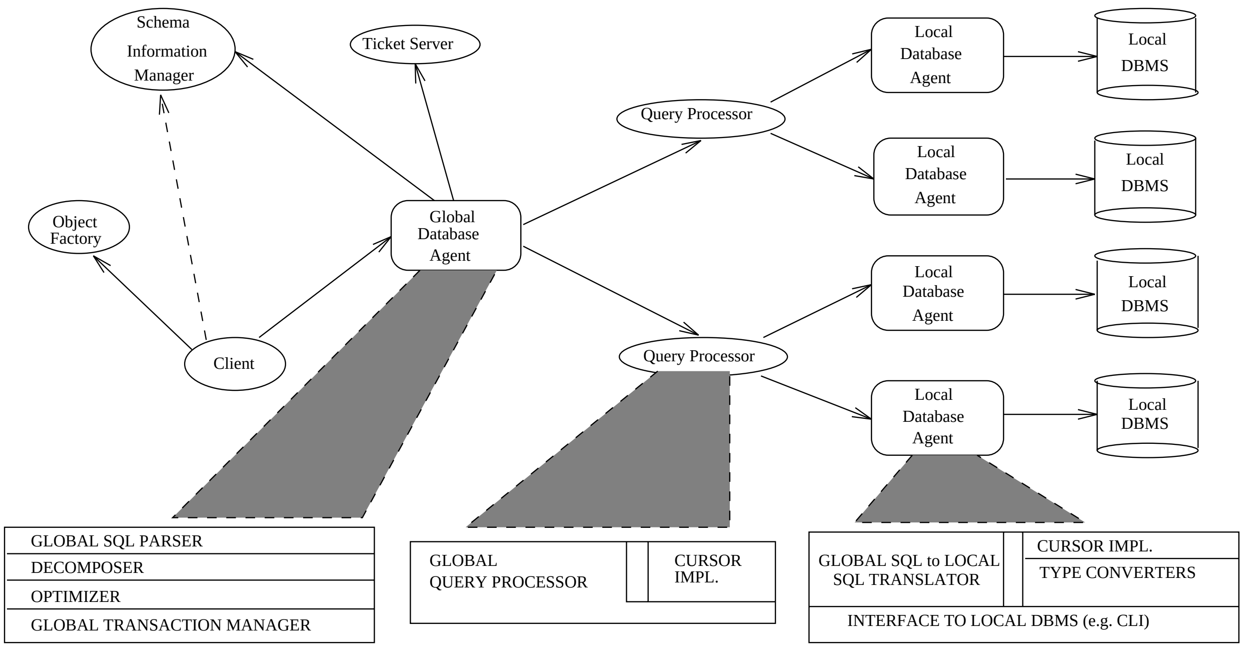


Figure 1: Architecture of MIND

4 SCHEMA INTEGRATION

In this section, we describe the schema integration process which constitutes the base of MIND query processing.

A four-level schema architecture is necessary to address the requirements of dealing with distribution, autonomy and heterogeneity in a multidatabase system. This schema architecture includes four different kinds of schemas:

1) Local Schema: A local schema is the schema managed by the local database management system. A local schema is expressed in the native data model of the local database and hence different local schemas may have different data models.

2) Export Schema: An export schema is derived by translating local schemas into a canonical data model. The process of schema translation from a local schema to an export schema generates mappings between the local schema objects and the export schema objects.

3) Global (Federated) Schema: A global schema combines the independent export schemas to a (set of) integrated schema(s). It also includes the information on data distribution (mappings) that is generated when integrating export schemas. The global query manager transforms queries on the global schema into a set of queries on one or more related export schemas.

4) External Schema: In addition, it should be possible to store additional information that is not derived from export databases. An external schema defines a schema for a user or an application. An external schema can be used to specify a subset of information in a global schema that is relevant to the users of the external schema. Additional integrity constraints can also be specified in the external schema.

The classes in export and global schemas behave like ordinary object classes. They consist of an interface and an implementation. But unlike ordinary classes, which store their objects directly, the implementations of the classes in these schemas derive their objects from the objects of other classes.

4.1 <u>Classification of Schema Conflicts</u>

The potential schematic conflicts between the export schemas are identified according to the following classification framework:

1) Semantic Conflicts (Extent Conflicts): Two designers do not perceive exactly the same set of real world objects. For instance, a 'student' class may appear in one schema, while a more restrictive 'cs-student' class is in another schema. This is a kind of conflict which relate to the extents of classes. The extent of a class is the set of objects in that class. According to their extents, relationships between classes are classified in four groups.

a) Identical Classes : Classes in export schemas represent the same set of objects (e.g. 'instructors' in one database and 'lecturers' in another database). These classes are merged into a single class in the global schema (e.g. g-instructors). If a global query is sent to the merged class, join or outerjoin operation need to be performed to obtain the final result, since there is no difference between join and outer join for this case. As an example, if a query is sent to the 'g-instructors' class, to obtain the final answer, the results returned from 'instructors' class and 'lecturers' class need to be joined (or outer joined).

b) Intersecting Classes: Classes may represent overlapping sets of objects. For instance, 'student' and 'employee' classes in two databases may have common objects which represent 'research-assistants'. Such classes are integrated in the global schema as classes having a common subclass that contains the set of common objects. If a global query is issued against to the new subclass (i.e research-assistants), then global join operation is necessary to obtain the result. If two semantically equivalent classes are intersecting classes, like two 'employee' classes in two different DBMSs, then in global schema these two classes can be represented as a single class, such as a 'g-employee' class. If a global query is issued against to 'g-employee' class, the global outer join operation, between the two local 'employee' classes, is necessary to obtain the final result.

c) Inclusion Classes: The extent of one class may be a subset of another class (e.g. Employee & Manager classes). In the global schema the more restrictive class is represented as the subclass of the other class. Assume that 'g-manager' is the subclass of 'g-employee' class in the global schema. If a query is sent to 'g-manager' class and if 'g-employee' has an information that does not exist in 'g_manager', then in order to obtain the result containing this missing information, a join operation is necessary between 'employee' and 'manager' classes.

d) Disjoint Classes: These are classes whose extents are completely different but related (e.g. Graduate vs Undergraduate students). Such classes are generalized into a superclass in the global schema (g-student). If a global query is sent to 'g-student' class, then union operation is necessary to merge the results of two local databases.

2) Structural Conflicts (Descriptive Conflicts): When describing related sets of real-world objects, two designers do not perceive exactly the same set of properties. This second kind of conflict includes naming conflicts due to the homonyms and synonyms, attribute domain, scale, constraints, operations etc. For instance, different number of attributes or methods may be defined for semantically equivalent classes; or different names may be used for identical attributes. Such conflicts are resolved by providing a mapping between the class and attribute names in the global schema and class and attribute names in export schemas. This mapping is expressed in the object definition language described in Section 4.2.

4.2 MIND Schema Integrator

In MIND, LDAOs translate the local schemas into the export schemas using ODL, and then their export schemas are stored in the SIMO. SIMO integrates these export schemas to generate a global schema. Schema integration is a two phase process:

1) Investigation phase: First commonalities and discrepancies among the export schemas are determined. Currently, this phase is manual. That is, the DBA examines export schemas and defines the applicable set of inter-schema correspondences. The basic idea is to evaluate some degree of similarity between two or more descriptions, mainly based on matching names, structures and constraints. The correspondences are identified according to the classification of schema conflicts given in Section 4.1.

2) Integration phase: The integrated schema is built according to the inter-schema correspondences. The integration phase cannot be fully automated. Interaction with the DBA is required to solve conflicts among export schemas. In MIND, the integration of export schemas is currently performed by using an object definition language (ODL) which is a specification language used to define the interfaces to object types that conform to the ODMG-93 object model. The DBA builds the integrated schema as a view over export schemas. The functionalities of ODL allow selection and restructuring of schema elements from existing local schemas. In ODL, a schema definition is a list of interface definitions whose general form is as follows:

```
interface classname:superclass_list {
     extent    extentname;
     key       attr1 [, attr2, ...];
     attribute attr_type attr1;
     ...
     relationship reltype relname
        inverse OtherClass::invrel;
     ...
}
```

where **classname** is the name of the class whose interface is defined; **superclass_list** includes all superclasses of the class; **extentname** provides access to the set of all instances of the class; **key** allows to define a set of attributes which uniquely identifies each object of the class; **attribute** and **relationship** constitute the essential information about the class. **relationship** names and defines a traversal path for a relationship. A traversal path definition includes designation of the target class and information about the inverse traversal path found in the target class (Cattell, 1994).

In addition to its interface definition, each class needs information to determine the extent and to map the attributes onto the local ones. The general syntax for this mapping definition which is similar to (Busse R. et al., 1994) is provided in Table 1.

```
mapping classname {
     origin  originname1: classname1 alias1 [,
             originname2: classname2 alias2, ...];
     def_ext  extname  as
              select
              from alias1, alias2, ...
              where alias1.attrname *=*
                    alias2.attrname [and ...];
     def_att attr1 as [alias1.attrname |
                         /* simple reference */
                   select alias1.attrname,
                          alias2.attrname
                   from alias1, alias2
                   where  ...; ]   /* query */
     def_rel relname as
           [select *
            from alias1, alias2 ...
            where ... ;];
}
```

Table 1. The Mapping Definition

The keyword **mapping** marks the block as a mapping definition for the derived class **classname**. The **origin** clauses define a set of private attributes that store the back-references to those objects, from which an instance of this class has been derived. The extent derivation clause starting with **def_ext** specifies a query that defines full instantiation of the derived class. Differentiation between join and outer join operations to resolve extent conflicts is similar to standard SQL, that is (=) is used for join and (*=) is used for outer join in the where predicate of the query in **def_ext** clause. A list of **def_att** lines defines the mapping for each attribute of the class. And finally, a set of **def_rel** lines express the relationships between derived classes as separate queries which actually represent the traversal path definitions.

After the global schema is obtained, SIMO provides the necessary information to the GQMO on demand. We are currently developing a graphical tool which will automatically generate textual specification of the global schema. Our ultimate aim is to establish a semi-automated technique for deriving an integrated schema from a set of assertions that state the inter-schema correspondences. The assertions will be derived as a result of the investigation phase. For each type of assertion, there will correspond an integration rule so that the system knows what to do to build the integrated schema.

5 QUERY DECOMPOSITION

In a multidatabase system, a query is expressed in the global query language against the global schema. Since the global query may need data from various local DBMSs, it is necessary to decompose the global query into subqueries such that data needed by each subquery are available from one local DBMS. After the results are returned by LDAOs, they should be combined into the answer by post-processing operations such as join, projection, selection, union etc.

When a global query is issued to the system, it is sent to the query decomposer. The global query decomposition process is highly dependent on the schema integration information. In the following, we will provide insight to the query decomposition process through an example. The complete query decomposition algorithm is provided in Appendix 1.

Example 1 :
Consider the following global query and the schemas given in Figures 2 and 3.

```
q:   select e.name, e.age, d.dept_name
     from employee e, department d
     where e.salary > 3000 and
           e.dept_no = d.dept_no and
           d.address = 'ODTU/Ankara';
```

Note that the global schema is obtained through integration of three export schemas and the query provided in this example requires data from all of these databases. Then, we have three subqueries each of which will be sent to a different local DBMS.

First step is to decompose the **from** clause. For both class references, that is, employee and department, the origin list is extracted from the mapping definition of these classes. As an example, for employee class, the origin list (Figure 3b) is:

```
origin      db1:emp e1
origin      db1:mgr_emp m1
origin      db2:employee e2
origin      db3:employee e3
origin      db2:division d2
```

All class references in the above origin list are inserted into the **from** clauses of the corresponding subqueries. In the second step, projection list is processed. For each of the attributes (name, age, and dept_name), attribute definitions are obtained from the mapping of the related class. As an example, attribute definition includes the following query for *name* attribute of employee class:

```
q1:   select e1.ename, e2.name, e3.ename
      from e1, e2, e3
```

While forming the projection lists of subqueries, the attribute in the projection list of q_1 is used rather than the global attribute. Therefore, the global attribute *name* is inserted into the projection list of the first subquery as *ename* and so on. After all attributes in the projection list are processed and inserted into their corresponding subqueries in the same way, search conditions in the **where** predicate of the global query is processed. While decomposing a search condition, the right hand side (RHS) of the predicate may contain a constant or an expression. If the RHS of the predicate is an expression, then four cases need to be considered. The attribute definition of the left hand side (LHS) may contain a simple reference (Table 1) and RHS may contain a query, or vice versa. Or both may contain simple references and lastly both may contain queries in the attribute definitions. Each of these four cases are handled differently by the decomposition algorithm given in Appendix 1. If the attribute definition of any side contains a query, the where predicate of this query, if exists, also needs to be decomposed.

In the example, to process the first predicate (e.salary > 3000), the attribute definition of *salary* is obtained from the mapping definition (Figure 3b) as follows :

```
q2:   select e1.salary, e2.salary, e3.salary
      from e1, e2, e3;
```

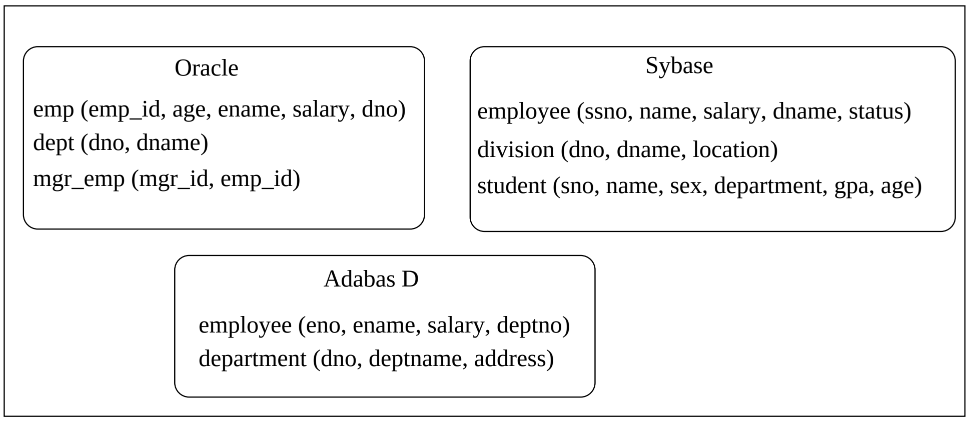
The predicates of subqueries are formed using the attributes listed in the projection list of q_2. The predicate for the first subquery is *e1.salary > 3000*, and *e2.salary > 3000* is inserted into the second subquery etc.

For the second predicate, attribute definitions of both sides are obtained. For, *dept_no* attribute of employee class, the following query is given in the mapping definition (Figure 3b):

```
q3:   select e1.dno, d2.dno, e3.deptno
      from e1, e2, e3, d2
      where e2.dname = d2.dname;
```

Attribute definition of *dept_no* attribute of department class (Figure 3c), on the other hand, contains the query given below:

```
q4:  select d1.dno, d2.dno, d3.dno
     from d1, d2, d3;
```



Oracle
emp (emp_id, age, ename, salary, dno)
dept (dno, dname)
mgr_emp (mgr_id, emp_id)
Sybase
employee (ssno, name, salary, dname, status)
division (dno, dname, location)
student (sno, name, sex, department, gpa, age)
Adabas D
employee (eno, ename, salary, deptno)
department (dno, deptname, address)

Figure 2: Local Databases

The global join predicate (e.dept_no = d.dept_no) is converted into the predicate (e1.dno = d1.dno) for the first subquery. For the above case, the where predicate of the query given in q_3 should also be decomposed. For the example, (e2.dname = d2.dname) is inserted into the second subquery. Finally, the predicate (e3.deptno = d3.dno) is inserted into the third subquery.

During this decomposition process, the attributes in the where predicate of the global query are also inserted into the projection lists of the generated subqueries. The justification of this step becomes clear in Section 6.

The final three subqueries generated are:

-
  ```
  select e1.emp_id, e1.ename, e1.age,
         e1.salary, d1.dname
  from   emp e1, dept d1
  where  e1.salary > 3000 and
         e1.dno = d1.dno;
  ```

-
  ```
  select e2.ssno, e2.name, e2.salary,
         d2.dname, d2.location
  from   employee e2, division d2
  where  e2.salary > 3000 and
         d2.location = 'ODTU/Ankara' and
         e2.dname = d2.dname;
  ```

-
  ```
  select e3.eno, e3.ename, e3.salary,
         d3.deptname, d3.address
  from   employee e3, department d3
  where  e3.salary > 3000 and
         e3.deptno = d3.dno and
         d3.address = 'ODTU/Ankara';
  ```

  □

While decomposing the global query into subqueries, the operations that need to be performed to merge the results of these subqueries are also determined. For this purpose the extent definitions of the classes referenced in the global query are used. These extent definitions either have where predicates containing the join attributes and the operation to be performed (join, outer join), or do not contain any where condition implying the case of distinct classes and the operation to be performed is union.

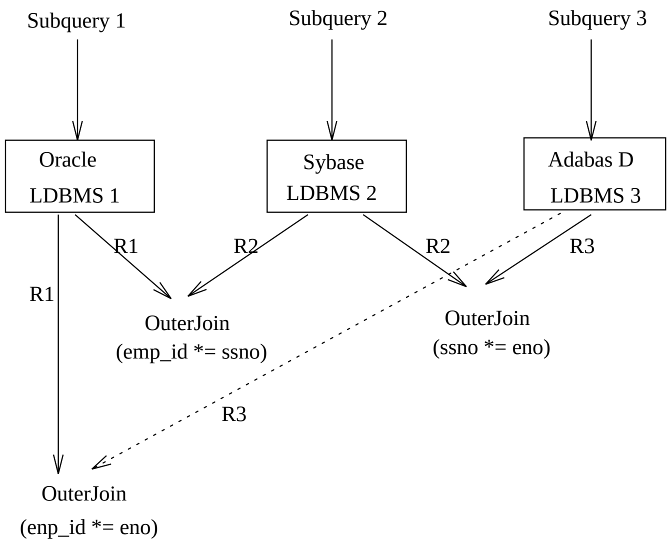


Figure 4: Join List of Example 1

The operations to be performed to combine the results of the subqueries generated for Example 1 is shown in Figure 4.

In decomposing a query, the following observation is made: If a query is decomposed based only on the selection criteria, in other words, if the subqueries are sent only to those sites that contain the attributes in the selection predicate, information loss may occur. As an example, consider a database, DB1, and assume that a query is given which contains a selection criterion on an attribute that does not exist in the classes of DB1. In such a case, it seems as if the query should not be sent to DB1. However, if the projection list of this global query contains an

Global Schema

```
employee (ssno, name, age, salary, dept_no, manager, status)
department (dept_no, dept_name, address)
student (stu_id, sname, gpa, sex, dept_name, age)
assistants (asst_id, name, gpa, stu_dept, sex, salary, emp_dno)
```

(a)

Mapping Definitions

```
mapping employee {
  origin  DB1:emp e1,
          DB1:mgr_emp m1,
          DB2:employee e2,
          DB2:division   d2,
          DB3:employee e3;
  def_ext  emp_ext as
     select * from e1, e2, e3 where e1.emp_id *= e2,ssno and
                                     e2.ssno *= e3.eno and
                                     e1.emp_id *= e3.eno;
  def_att ssno as
     select e1.emp_id, e2.ssno, e3.eno from e1,e2,e3;
  def_att name as
     select e1.ename, e2.name, e3.ename from e1, e2, e3;
  def_att age as e1.age;
  def_att salary as
     select e1.salary, e2.salary, e3.salary from e1, e2, e3;
  def_att dept_no as
     select e1.dno, d2.dno, e3,dno from e1, e2, d2, e3
     where e2.dname = d2.dname;
  def_att manager as
     select m1.mgr_id from e1, m1 where e1.emp_id = m1.emp_id;
  def_att status as e2.status;
}
```

(b)

```
mapping department {
  origin DB1:dept d1,
         DB2:division d2,
         DB3:department d3;
  def_ext dept_ext as
     select * from d1,d2,d3 where d1.dno*=d2.dno and
              d2.dno*=d3.dno and d1.dno*=d3.dno;
  def_att dept_no as
     select d1.dno, d2.dno, d3.dno from d1,d2,d3;
  def_att dept_name as
     select d1.dname, d2.dname, d3.deptname
     from d1,d2,d3;
  def_att addrress as
     select d2.location, d3.address from d2, d3;
}
```

(c)

```
mapping student {
  origin DB2:student s1;
  def_ext stu_ext as select * from s1;
  def_att stu_id as s1.sno;
  def_att sname as s1.name;
  def_att gpa as s1.gpa;
  def_att sex as s1.sex;
  def_att dept_name as s1.department;
  def_att age as s1.age;
}
```

(d)

```
mapping assistants {
  origin DB2:student s1,
         DB1:emp e1,
         DB2:employee e2,
         DB2:division d2,
         DB3:employee e3;
  def_ext asst_ext as
     select * from s1,e1,e2,e3
     where s1.sno=e2.ssno and s1.sno=e2.ssno
                         and s1.sno=e3.eno;
  def_att asst_id as
     select s1.sno, e1.emp_id, e3.eno from s1,e1,e3;
  def_att name as
    select s1.sname, e1.ename, e3.ename from s1,e1,e3;
  def_att gpa as s1.gpa;
  def_att stu_dept as s1.department;
  def_att sex as s1.sex;
  def_att salary as
     select e1.salary, e2.salary,e3.salary from e1,e2,e3;
  def_att emp_dno as
     select e1.dno, d2.dno,e3.dno from e1,e2,d2,e3
     where e2.dname=d2.dname;
}
```

(e)

Figure 3: Global Schema and Mappings

attribute available only from DB1, then it is also necessary to send this query to DB1. In this way it is possible to obtain the attribute, which exist only at DB1, of a tuple that satisfies the selection criterion. Yet with this technique, invalid tuples may appear in the final result since the tuples collected from DB1 may not satisfy the selection criterion. Therefore the final result is processed to eliminate invalid tuples as explained in Section 6.

In Example 1, the global query contains a selection on the *address* attribute but employee class in the first local DBMS does not have such an attribute. However, since *age* attribute that is in the projection list of the global query, exists only in this database, a subquery is generated for this local DBMS to make it possible to extract the age information for employees selected from the second and the third local DBMSs.

6 PROCESSING PARTIAL RESULTS

After a global query is decomposed, it is executed according to the execution plan dictated by the global query optimizer. Within the scope of MIND, two query optimization techniques have been developed.

In (Evrendilek, C. et al., 1995b; Evrendilek, 1995; Evrendilek, C. et al., 1995a), a cost based query optimization strategy is explained. In this strategy, the optimization algorithm tries to maximize the parallelism in execution while taking the federated nature of the problem into account. In optimizing intersite joins, a two-step heuristic algorithm is proposed which incorporates parameters stemming from the nature of multidatabases such as the time taken by global subqueries at the local DBMSs, transmission overhead incurred while communicating intermediate results during intersite join processing and the cost of converting global subquery results into the canonical data model. In this technique, an execution plan is generated before sending the subqueries to the local DBMSs. In the second version of MIND query optimizer, however, a dynamic optimization is used as explained in (Ozcan, 1996; Dogac, A. et al., 1996). The main idea behind dynamic query processing is to handle the uncertainty present in the system due to the local loads of the autonomous LDBMSs, by using the actual appearance times instead of estimations. It also tries to exploit the inherent parallelism in the system as much as possible. In this technique, the intersite operations are scheduled at run-time.

After the subqueries are executed according to the plan dictated by the query optimizer, it is necessary to merge the results returned from local DBMSs to form the final answer. This merging is done through post-processing operations, namely join, outer join or union. The last result obtained through intersite operations is processed to eliminate invalid information. Finally, the result is projected onto the projection list of the global query.

In the following, an example is provided to clarify this process.

Example 2: Assume, the following execution plan is dictated by the global query optimizer for the results of subqueries of Example 1:

$$((R_1 \Leftrightarrow R_2) \Leftrightarrow R_3)$$

where R$_i$ refers to the result of the subquery that is sent to the local database i, $\Leftrightarrow$ implies any intersite operation and the paranthesization shows the execution order. Three partial results are returned as shown in Figure 5.

First, R$_1$ and R$_2$ are merged via outer join operation as shown in Figure 4. As the result of this operation, the intermediate result shown in Figure 6 is obtained. While performing an outer join operation, a "NotApplicable" value is inserted for the attribute that does not appear in the corresponding database.

Next, R$_3$ is combined with the first intermediate result through outer join, which yields in the result given in Figure 7. Note, however, that the tuple having the name 'Ali Ozturk' which should not be in the final answer is selected as a qualified tuple during outerjoin operations, since the first local database do not have the address information and selection could not be performed there. Yet it is necessary to send a subquery to this site to obtain the age field that only exist in this site. Thus, there may be tuples that do not satisfy the selection criteria. In order to prevent such invalid tuples, final intermediate result is further processed to eliminate tuples having "NotApplicable" value for one of the attributes that exist in the **where** predicate of the global query. To perform such an elimination, the attributes which appear in the **where** predicate of the global query should also be collected from local DBMSs, by including them into the projection list, to enable checking for "NotApplicable" values of these attributes. In Example 1, the attributes in the where clause are inserted into the projection lists of subqueries for this reason. In this example, 'Ali Ozturk' has "NotApplicable" values for *location* and *address* attributes, both representing the same information with global *address* attribute, and therefore it is eliminated from the result.

Finally, it is necessary to project the result into the projection list of the global query. During projection operation, synonymous attributes that represent the same entity are extracted and eliminated from the result. For example, *ename, name, ename* are semantically the same, thus two of them are eliminated. The attribute having a value other than "NotApplicable" is preferred to be preserved in the result while eliminating the others. The final result is as shown in Figure 8. □

7 CONCLUSIONS AND THE FUTURE WORK

In this paper, schema integration and query decomposition processes in a multidatabase environment have been addressed. It is observed that query decomposition is highly dependent on schema integration and thus the two problems should be considered together. In this context, an algorithm for query decomposition process is provided as it is implemented in MIND project.

Data inconsistency is another important factor that affects the complexity of query processing. When a real-world object is represented by different entities in local databases, data inconsistency may occur. For example, an employee having an age of 24 in one local DBMS, may have the age value as 27 in another local DBMS. When these two employee classes are integrated

emp_id	ename	age	salary	dname
00123	Asuman Dogac	25	10000	computer
02234	Sena Nural	23	4000	research
12397	Fatma Ozcan	24	5000	computer
23476	Ali Ozturk	40	3500	accounting

(a)

ssno	name	salary	dname	location
00123	Asuman Dogac	10000	computer	ODTU/Ankara
02234	Sena Nural	4000	research	ODTU/Ankara
45800	Pinar Koksal	5000	research	ODTU/Ankara

(b)

eno	name	salary	deptname	address
00123	Asuman Dogac	10000	computer	ODTU/Ankara
12397	Fatma Ozcan	5000	computer	ODTU/Ankara

(c)

Figure 5: Partial Results (a) form ORACLE database (b) from SYBASE database (c) f rom ADABAS database

emp_id	ename	age	salary	dname	ssno	name	salary	dname	location
00123	Asuman Dogac	25	10000	computer	00123	Asuman Dogac	10000	computer	ODTU/Ankara
02234	Sena Nural	23	4000	research	02234	Sena Nural	4000	research	ODTU/Ankara
12397	Fatma Ozcan	24	5000	computer	NotApp	NotApp	NotApp	NotApp	NotApp
23476	Ali Ozturk	40	3500	accounting	NotApp	NotApp	NotApp	NotApp	NotApp
NotApp	NotApp	NotApp	NotApp	research	45800	Pinar Koksal	5000	research	ODTU/Ankara

Figure 6: First intermediate result

emp_id	ename	age	salary	dname	ssno	name	salary	dname	location	eno	name	salary	deptname	address
00123	Asuman Dogac	25	10000	computer	00123	Asuman Dogac	10000	computer	ODTU/Ankara	00123	Asuman Dogac	10000	computer	ODTU/Ankara
02234	Sena Nural	23	4000	research	02234	Sena Nural	4000	research	ODTU/Ankara	NotApp	NotApp	NotApp	NotApp	NotApp
12397	Fatma Ozcan	24	5000	computer	NotApp	NotApp	NotApp	NotApp	NotApp	12397	Fatma Ozcan	5000	computer	ODTU/Ankara
23476	Ali Ozturk	40	3500	accounting	NotApp	NotApp	NotApp	NotApp	NotApp	NotApp	NotApp	NotApp	NotApp	NotApp
NotApp	NotApp	NotApp	NotApp	NotApp	45800	Pinar Koksal	5000	research	ODTU/Ankara	NotApp	NotApp	NotApp	NotApp	NotApp

Figure 7: Second Intermediate Result

name	age	dept_name
Asuman Dogac	25	computer
Sena Nural	23	research
Fatma Ozcan	24	computer
Pinar Koksal		research

Figure 8: Final Result

to obtain a global schema, data inconsistency occurs on age attribute of this employee. To resolve data inconsistencies among databases, aggregation functions (Dayal, 1983) may be used.

We propose to use aggregation function definitions for each attribute in the global schema by extending the mapping definition to handle these functions. While deriving a global schema attribute from export schemas, aggregation type is to be determined and the mapping definition is to be formed accordingly.

Modification of the query processing algorithm presented in this paper, to handle data inconsistency is explained in (Nural, 1995).

References

Busse R., Frankhauser P., and Neuhold E. J. (1994). Federated schemata in ODMG. In *East/West Database Workshop*, Klagenfurt.

Cattell, R. (1994). *The Object Database Standard: ODMG-93*. Morgan Kaufmann.

Dayal, U. (1983). Processing queries over generalization hierarchies in a multidatabase system. In *International Conference on Very Large Data Bases*.

Dayal, U. (1985). Query Processing in a Multidatabase System. In Kim, W., editor, *Query Processing: Database Systems*, pages 81–108. Springer-Verlag, New York.

DEC (1994). *ObjectBroker, System Integrator's Guide*. Digital Equipment Coop.

Dogac, A. e. a. (1994). METU Object-Oriented Database System, Demo Description. In *Proceedings of ACM SIGMOD Intl. Conf. on Management of Data*, Minneapolis.

Dogac, A., Altinel, M., Ozkan, C., and Durusoy, I (1995a). Implementation Aspects of an Object-Oriented DBMS. *ACM Sigmod Record*, 24(1).

Dogac, A., Dengi, C., Kilic, E., Ozhan, G., Ozca n, F., Nural, S., Evrendilek, C., Halici, B., a nd Koksal, P., A., and Mancuhan, S. (1996). A Multidatabase System Implementation on CORBA. In *6th Intl. Workshop on Research Issues in Data Engineering (RIDE-NDS '96)*, New Orleans.

Dogac, A., Dengi, C., Kilic, E., Ozhan, G., Ozcan, F., Nural, S., Evrendilek, C., Halici, B., Arpinar, B., Koksal, P., Kesim, N., and Mancuhan, S. (1995b). METU Interoperable Database System. *ACM Sigmod Record*, 24(3).

Dogac, A., Dengi, C., Kilic, E., Ozhan, G., Ozcan, F., Nural, S., Evrendilek, C., Halici, B., Arpinar, B., Koksal, P., Kesim, N., and Mancuhan, S. (1995c). A multidatabase system implementation on CORBA, demo description. In *Intl. Conf. on OOPSLA'95*, Austin, Texas.

Evrendilek, C. (1995). *Multidatabase Query Processing and Optimization*. PhD thesis, Dept. of Computer Engineering, Middle East Technical University. in preparation.

Evrendilek, C., Dogac, A., Nural, S., and Ozcan, F. (1995a). Multidatabase Query Optimization. Technical report, Software R&D Center of TUBITAK, Middle East technical University.

Evrendilek, C., Dogac, A., Nural, S., and Ozcan, F. (1995b). Query decomposition, optimization and processing in multidatabase systems. In *Proc. of Workshop on Next Generation Information Technologies and Systems*, Naharia, Israel.

Finance, B., Smahi, V., and Fessy, J. (1995). Query Processing in IRO-DB. In *Proc. of 4th Intl. Conf. on Deductive and Oject-Oriented Databases (DOOD 95)*, Singapore.

Kilic, E., Ozhan, G., Dengi, C., Kesim, N, Koksal, P., and Dogac, A. (1995). Experiences in Using CORBA in a Multidatabase Implementation. In *Proc. of 6th Intl. Workshop on Database and Expert System Applications*, London.

Kim, W. (1995). *Modern Database Systems*. Addison-Wesley.

Meng, W. and Yu, C. (1995). Query processing in multidatabase systems. In Kim, W., editor, *Modern Database Systems*. ACM Press.

Nural, S. (1995). Query Processing in Multidatabase Systems. Master's thesis, Dept. of Computer Engineering, METU. in preparation.

Ozcan, F. (1996). Dynamic Query Optimization in a Distributed Object Management Platform. Master's thesis, Dept. of Computer Engineering, METU.

Scheurmann, P. and Chong, E. (1994). Role-based Query Processing in Multidatabase Systems. In *Proc. of EDBT*.

Soley, R. M. (1992). *Object Management Architecture Guide*. OMG, second edition.

Appendix 1

A select clause is simply represented as :

```
select  <select_list>
from    <class_reference_list>
where   <search_conds>
```

where

```
<select_list>          ::= <expr> [, <expr> ...]
<class_reference_list> ::= <class_name> <alias>
              [, <class_name> <alias> ...]
<search_conds>         ::= <search_cond>
                           [AND <search_cond> ...]
<search_cond>          ::= <left> <operator> <right>
<expr>                 ::= <alias>'.'<attr_name>
```

In the above representation both <left> and <right> are expressions (< expr>) and <operator> can be any operator such as '=', '>', '<=' etc.

In the light of this structure, decomposition process is given in Algorithm 1.

Algorithm 1 : Decomposition Process
Input : Global Query
Output : Subqueries

```
for all class_references i          /* of Global Query */
    class_info = get_class_info (class_name (i));
    for all origin definitions j      /* of class_info */
        k = find_database (alias (j));
        add_to_class_reference (k);
    ext_query = get_def_ext(class_name (i));
        /*get extent definition */
    for all search_conds m                 /* of ext_query */
        db1 = find_database (right(m).alias);
        db2 = find_database (left(m).alias);
        decide_operation_type(operator(m));
        add_to_join_list(db1, db2);
for all select_items i                /* of Global Query */
    info = get_attr_info (expr (i));
    if (info.type = 'query') /* there exist a select clause */
        for all select_items j          /* of info */
            k = find_database (alias (j));
            add_to_select_list (k);
        decompose_where (info);                   /* Algorithm 2 */
    else                  /* there exist a simple reference */
        k = find_database (alias)
        add_to_select_list(k);
for all search_conds i            /* of Global Query */
    if (right(i).type <> 'constant')
        info1 = get_attr_info (right (i));
        info2 = get_attr_info (left (i));
        if (info1.type = 'query') and (info2.type = 'query')
            for all select_items m                /* of info1 */
                for all select_items l            /* of info2 */
                    db1 = find_database (alias (m));
                    db2 = find_database (alias (l));
                    if (db1 = db2)
                        add_to_where_pred (db1);
                    else
                        add_to_join_list (db1, db2);
                decompose_where (info2);
            decompose_where (info1);
        else if (info1.type = 'query')
            and (info2.type = 'simple_ref')
            db2 = find_database(info2.alias);
            for all select_items m                 /* of info1 */
                db1 = find_database(alias(m));
                if (db1 = db2)
                    add_to_where_pred(db1);
            decompose_where(info1);
        else if (info1.type = 'simple_ref')
            and (info2.type = 'query')
            db1 = find_database(info1.alias);
            for all select_items m                 /* of info2 */
                db2 = find_database(alias(m));
                if (db1 = db2)
                    add_to_where_pred(db1);
            decompose_where(info2);
        else if (info1.type = 'simple_ref')
            and (info2.type = 'simple_ref')
            db1 = find_database(info1.alias);
            db2 = find_database(info2.alias);
            if (db1 = db2)
                add_to_where_pred(db1);
            else
                add_to_join_list(db1,db2);
    else                                /* right is constant */
        info = get_attr_info (left (i));
        if (info.type = 'query')
            for all select_list items j          /* of info */
                k = find_database (alias (j));
                add_to_where_pred (k);
            decompose_where (info);
        else
            k = find_database (alias);
            add_to_where_pred (k);
```

Algorithm 2 : Decompose_Where
Input : a query (Given in def_attr part of mapping)

```
for all search conds i
    db1 = find_database (left (i));
    db2 = find_database (right(i));
    if (db1 = db2)
        add_to_where_pred (db1);
    else
        add_to_join_list(db1,db2);
```

Explanations of the functions used in the above algorithms :

add_to_reference_list (i): adds the reference item to the <class _reference_list> of ith subquery, that is, the subquery which will be sent to the ith local DBMS.
add_to_select_list (i): adds the expression formed using information obtained from the schema information to the <select_ list> of ith subquery.
add_to_where_pred (i): adds a search condition to the <search _conds> of ith subquery.
find_database(alias): it finds which database the class (referred with alias) resides on.
get_class_info(class_name): gets the origin list definitions of the mapping information of the given class. Returns the list of origin names, class names and aliases.
get_attr_info(expr): gets the attribute information (def_attr part of mapping definition). It returns a simple reference or a query and sets its type correspondingly.
get_def_ext(class_name): gets extent definition from the mapping definition. It returns the query given in def_ext part of the mapping.
add_to_join_list(db1, db2): adds a join condition between the two subqueries that are sent to the local DBMS 'db1' and the local DBMS 'db2'.
decide_operation_type(operator): decides the type of the intersite operation to be performed between the results of subqueries. It returns OuterJoin, Join or Union according to the *operator*. If the *operator* is *=, returns OuterJoin, if it is =, returns Join and finally if def_ext part do not contain a where predicate it returns Union type. (These types will be considered during post-processing operations on the results of subqueries)