In: Cadlab-Report 3/94, Paderborn, 1994.

Object Migration in Federated Database Systems

Elke Radeke
Cadlab
Cooperation University of Paderborn & SNI AG
Bahnhofstr. 32, 33102 Paderborn, Germany
elke@cadlab.de

Marc H. Scholl
University of Ulm
Faculty of Computer Science
89069 Ulm, Germany
scholl@informatik.uni-ulm.de

Abstract

Existing federated database systems (FDBS) provide uniform access to multiple heterogeneous DBS, but do not enable objects to move across the DBS while retaining global identity. To fulfill this requirement of industrial users, we incorporate into FDBS a flexible object migration mechanism to move or duplicate objects from one component database system to another. We present different capabilities for "what" and "how" to migrate and consider the migration of both single objects and whole object graphs, i.e. an object together with related objects.

1 Introduction

In many enterprises, multiple different database systems have been built or acquired over the last several years. When data communication became easier, the request for coupling databases increased to allow controlled sharing of data. The approach of federated database systems (FDBS) [8, 16] became very important in this field because it retains the autonomy of the multiple existing database systems. The gigabytes of data inside the databases are retained and the big set of self-implemented or bought database applications continue running without any recoding. In addition, a uniform access is provided to all heterogeneous component databases by an additional software layer, the global FDBS interface.

Recently, research in this domain mainly concentrated on offering uniform data access, investigating issues such as database and schema integration, global query processing, and global transactions. But to the best of our knowledge, no existing FDBS product, prototype, or project (e.g. Pegasus [2], Mermaid [18], Carnot [22]) focus on a migration of data from one component database system (CDBS) to another.
Nevertheless, a wide spectrum of applications would benefit from a mechanism enabling migration of data (objects) within an FDBS. It may either be used as FDBS internal service or as external service for FDBS applications. Object migration as FDBS external service was requested by industrial users. They suffer from the huge amount of data (base) management systems in their enterprise and declare the controlled reduction of heterogeneity as a global enterprise goal. In [15] we derived the need to couple the database systems to

an FDBS and to incorporate object migration facilities into the FDBS. Object migration supports the stepwise elimination of some (legacy) data management system and moves still required data under FDBS control to another component DBS. If it retains global identity of migrated objects, existing global FDBS applications do not have to be recoded.
Object migration as internal service supports, for example, the strategy of [1] to execute global transactions locally in a DBS and migrate all necessary objects to this DBS. Furthermore, object migration is required if the global FDBS interface provides operations to flexibly change an object's class[1]. In case source and target class are mapped to different CDBS, the corresponding data have to be migrated from the source CDBS to the target CDBS and global identity has to be preserved. Consider, for example, an FDBS containing data of university persons where students' data are stored in *StudentDBS* and employees' data in *EmployeeDBS*. A university person may change from status (class) student to employee after his study. So, corresponding data have to be migrated from *StudentDBS* to *EmployeeDBS*, e.g. to make them available for local *EmployeeDBS* applications, but his global identity remains the same.

In this paper, we come up with a flexible solution for object migration in an FDBS. It allows to move, replicate, and copy (parts of) objects from one component database system to another in different degrees. Object movement and replication retain global identity of an object.

In contrast to some gateway database systems, e.g. Oracle, that offer the importation of foreign data from specific other DBS, we allow to migrate objects among arbitrary database systems within an FDBS, independent of whether they possess a gateway feature or not. Additionally, the FDBS approach allows to globally control the migrated objects, e.g. to guarantee data consistency between replicas, while gateway systems do not control redundancy between connected database systems. Hence, the risk for data inconsistency decreases. Since our approach may also preserve global identity of migrated objects, existing global FDBS applications do not have to be recoded when some object moves from one CDBS to another. On the other hand, gateway systems do not provide object identity spanning over all connected database systems.
Similar to gateway systems, tools supporting the migration from one isolated database system to another [4, 13] are also restricted to a given set of database systems, in this case only two. So, a flexible object migration across all database systems of an enterprise is quite difficult. It requires different migration tools and, hence, often different techniques. In contrast, our FDBS approach offers a uniform object migration mechanisms to migrate data among multiple database systems.
In some distributed database systems there are also mechanisms to move or replicate data from one site to another [14, 5]. But the purpose is different to that in FDBS. In distributed database systems, object migration is realized automatically according to access statistics in order to speed up data access. Such an automatic change of object locality, in general, is not desirable in FDBS. It could eliminate objects from a CDBS which are still required by some applications of the autonomous CDBS. Instead we allow to move, replicate, and copy objects on user demand to another CDBS. Nevertheless, we adopted migration concepts from distributed DBS, e.g. the feasibility to deeply migrate complex objects [5]. Due to a different

[1] A class embraces some objects of a type.

architecture of FDBS which also considers autonomy, some more differentiated concepts are required. For example, we distinguish different kinds of what data to migrate of an object by some object kinds and migration degrees.

The rest of this paper describes the concepts and sketches algorithms of our object migration mechanism. They are illustrated by an FDBS example out of the world of Computer Aided Concurrent Engineering (CACE).

2 Object Migration Concepts

In this section, we present concepts for a flexible object migration within an FDBS. Starting with a base model, we then provide different kinds of "what" and "how" to migrate and also specify corresponding migration operations for the global FDBS interface.

2.1 Base Model for Object Migration

Object migration in FDBS means that data are transferred from one component database system of the FDBS to another. Users initiate object migration by an operation of the FDBS global interface. Therefore they have to specify the (global) object and, in general, source as well as target CDBS.[2] An example for object migration is given in Fig. 1.

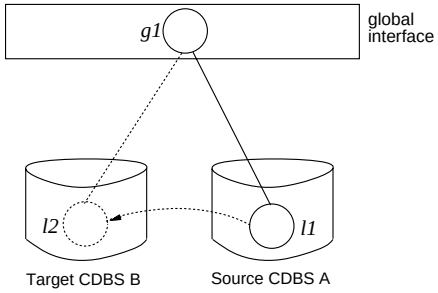

Figure 1: Object migration from CDBS A to B

Here, global object *g1* that was stored in CDBS A as local object *l1* shall be migrated through the global FDBS interface to CDBS B. We also see that object migration requires additional capabilities for object identification. In current FDBS, an association is provided between the (logical) global objects and their (physical) local objects in the CDBS [6, 7] in order to globally identify the objects of the various CDBS. To support object migration, this association of global and local objects must be dynamically updatable. In the example of Fig. 1 global object *g1* was first assigned to the local object *l1* and after object migration also/instead to local object *l2*.

Which data are migrated together with the object depends on the canonical data model of the FDBS and the object definition within this data model. Our solution is based on an object-oriented data model, COMIC [9] which is very similar to the upcoming standard from ODMG [3] both in semantics and syntax of the interface (C++ oriented). The object notion of both models is as follows:

[2] In some cases also transparent object migration is possible, see below.

An object consists of several attributes that express its structure and owns some methods describing the object's behaviour. We distinguish data attributes and attributes representing relationships to other objects (including part-of relationships). Types are used to formally define the attributes and methods of objects.

Generally, the following data is transferred when migrating an object to a target CDBS:

- All *data attributes* are migrated.
- Attributes representing *relationships* to other objects can optionally be migrated. By default they are set to NULL in the target CDBS and the related objects are not migrated. Optionally, the user may migrate general and/or part-of relationships together with the related objects (see Section 2.3.3).
- *Methods* are only supported by a few DBS, namely object-oriented DBS. Therefore they are migrated only in few cases. Since all objects of the same type own the same methods, method migration is coupled with type migration (see below).
- The *type* of an object is migrated as a prerequisite, if there is no equivalent type in the target CDBS. This is the case if the global type of the object to be migrated is not mapped to any local type in the target CDBS. Moreover, as a form of closure constraint on schemata, all types used within this type definition are migrated, e.g. those defining a relationship.
  A type migration, in general, induces schema evolution within the FDBS. This reduces the autonomy of the CDBS. But since we only extend schemas, local applications of the CDBS are further executable without any modification. There are also some special cases which do not need any type migration at all (see below).

During object migration, data has to be transformed according to the FDBS architecture. Fig. 2 shows how an object migration is processed using the 5 level FDBS schema architecture of [16].

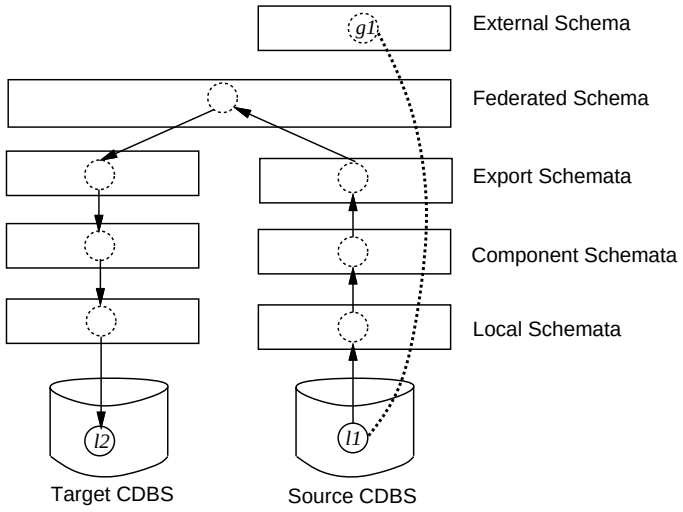

Figure 2: Object migration according to the 5 level FDBS architecture

Here, a global object *g1* that is accessed according to a specific external schema shall be migrated from a Source CDBS to a Target CDBS. Our object migration mechanism first

locates the corresponding local object(s) in the source CDBS. Then these local objects are transformed from the local schema of the source CDBS through a federated schema to the local schema of the target CDBS. Finally these data are stored into the TargetDBS as local object(s). This scenario is similar to reading data from one CDBS and writing it to another CDBS. But object migration is executed as one atomic global operation and it allows to retain the global identity of the migrated object (detailed differences see below).

To offer a flexible object migration mechanism we extend the base model in the following.

2.2 Migration Dimensions

Our object migration mechanism considers three migration dimensions (see Fig. 3):
1. *Object kind*: we classify different kinds of objects within an FDBS
2. *Migration degree*: different kinds of what data to be migrated with an object
3. *Operation primitive*: different kinds of how to migrate an object

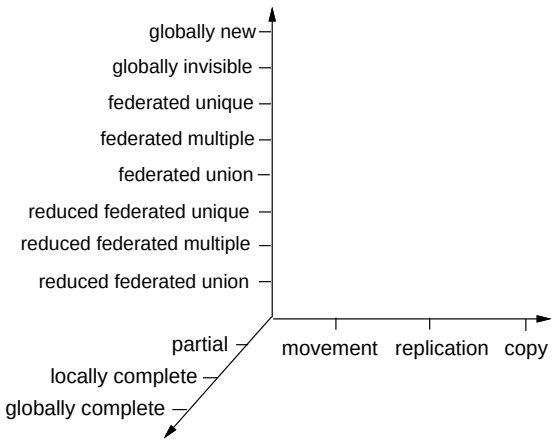

Figure 3: Object migration dimensions

We will first specify these dimensions in more detail and then illustrate them by an example.

2.2.1 Object Kind

As we have already mentioned in the object migration base model, all local objects of the various CDBS are mapped to (logical) global objects. Kent [10] recognized that multiple local objects (proxies) also may represent the same real world entity and thus map to a single global object. We analyzed the association of global objects to local objects more precisely and ended up with the following object kinds (see Fig. 4):

1. *Globally new*: global object not assigned to any local object of a CDBS, but stored in the Global-DB of the FDBS (used for global new and meta data).
2. *Globally invisible*: local object of some CDBS not assigned to any global object, i.e. not accessible via the global FDBS interface.
3. *Federated*: global object assigned to local objects of some CDBS without filtering any data.

(a) *unique*: global object assigned to exactly one local object of a CDBS
(b) *multiple*: global object assigned to equivalent local objects of multiple CDBS
(c) *union*: global object composed of some local objects of multiple CDBS or the Global-DB (possibly overlapping)

4. *Reduced federated*: similar to federated objects, but filtering is allowed (see filtering processor in [16]). That means not all data is visible in the global object of at least one assigned local object. We also distinguish reduced federated unique/multiple/union objects.

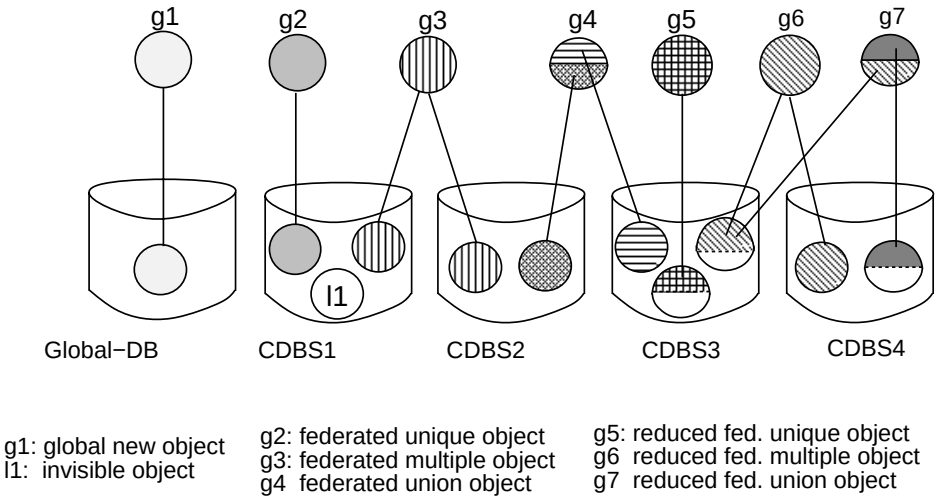

Figure 4: Object kinds in an FDBS

An object kind is a general characteristic within an FDBS, independent whether it possesses migration facilities or not. It is determined by the schema transformation of its type, i.e. how a local type of a CDBS is mapped to a global type in a federated/external schema. So, we derived the object kinds directly from the schema transformation processors which we already elaborated in [20]. For example, a reduced federated multiple object has a global type that was mapped by filtering and composition to some local types.
However, an object kind plays an important role for object migration. In conjunction with the migration degree, it determines what data of an object is migrated.

2.2.2 Migration Degree

The migration degree specifies how much of a global object shall be migrated. We distinguish three migration degrees. In Fig. 5 we illustrate them by migrating a reduced federated union object.

A) *Partial*: Globally visible data of a global object that is stored in a given source CDBS is migrated to a target CDBS.
In a variant A' - which eliminates the need for type migration in some cases (see below) - these data are migrated into a local object of the global type where data stored in different CDBS than Source CDBS are set to default values (e.g. NULL).

B) *Locally complete*: Both globally visible and invisible data is migrated from a single given source CDBS to a target CDBS.

C) *Globally complete*: All globally visible data of a global object is migrated from all CDBS to a target CDBS.

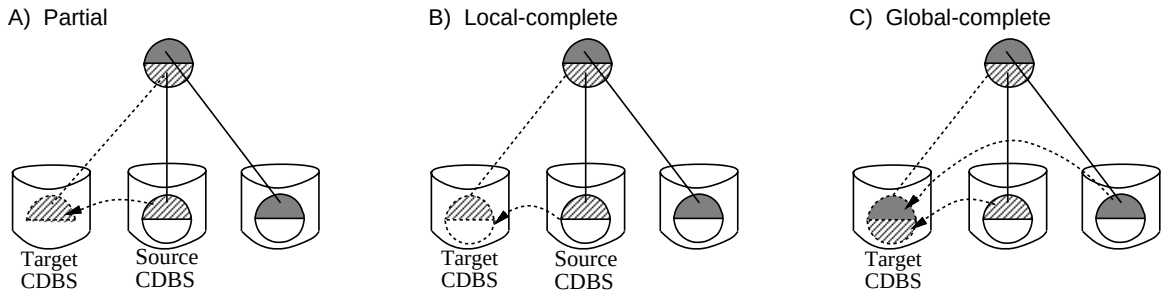

Figure 5: Migration degrees: A) partial, B) locally complete, C) globally complete

2.2.3 Operation Primitives

While the object kind and migration degree together determine *what* is migrated, the operation primitives as third dimension define *how* it is migrated. We offer three operation primitives that are illustrated in Fig. 6 for a reduced federated union object with globally complete migration degree.

1) *Absolute movement*: Objects are transferred into the target CDBS and deleted in the source CDBS, but object identity at the global interface remains the same.

2) *Replication*: Objects are transferred to the target CDBS but not deleted in the source CDBS. The duplicates have the same global object identity and the FDBS may guarantee data consistency between them.
In contrast to replicas in non-autonomous distributed DBS, FDBS are faced with the problem that local applications of a CDBS may change local objects without propagating the updates to replicas in the other CDBS. This can be solved by loosing the autonomy, using triggers [6], or allowing temporary inconsistency in the FDBS and, for example, specify time frames when data consistency should be checked [22].

3) *Independent copy*: Objects are also duplicated to the target CDBS, but get a different global object identity. Data consistency is not guaranteed between the duplicates.

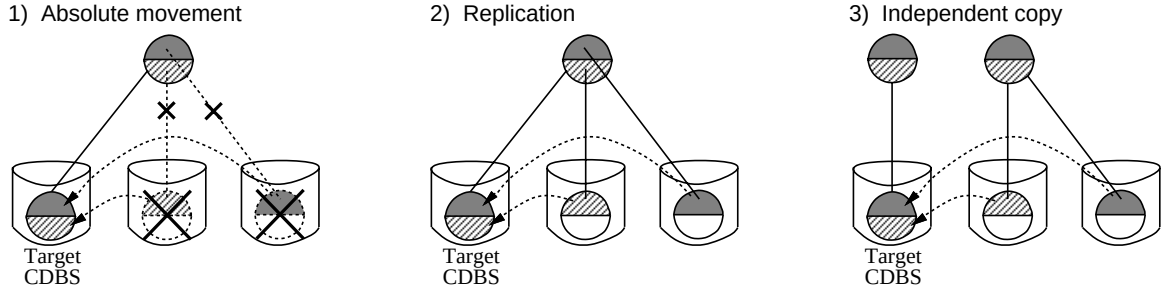

Figure 6: Operation Primitives: 1) absolute movement, 2) replication, 3) independent copy

If some data of an object has already been stored redundantly in source and target CDBS, e.g. for [reduced] federated multi objects, they are not transferred during object migration for absolute movement and replication. In particular absolute movement allows to decrease the redundancy within an FDBS which can be used during the reduction of the number of database systems [15].

2.2.4 Example

In order to illustrate the various migration dimensions, we present a concrete FDBS example in this subsection. It is a simplified scenario of one of our industrial project partner and originates from Computer Aided Concurrent Engineering (CACE) [21].
The CACE FDBS consists of three component database systems, one for each department of the enterprise. They are: DesignDBS, ProductionDBS and SalesDBS (Fig. 7).

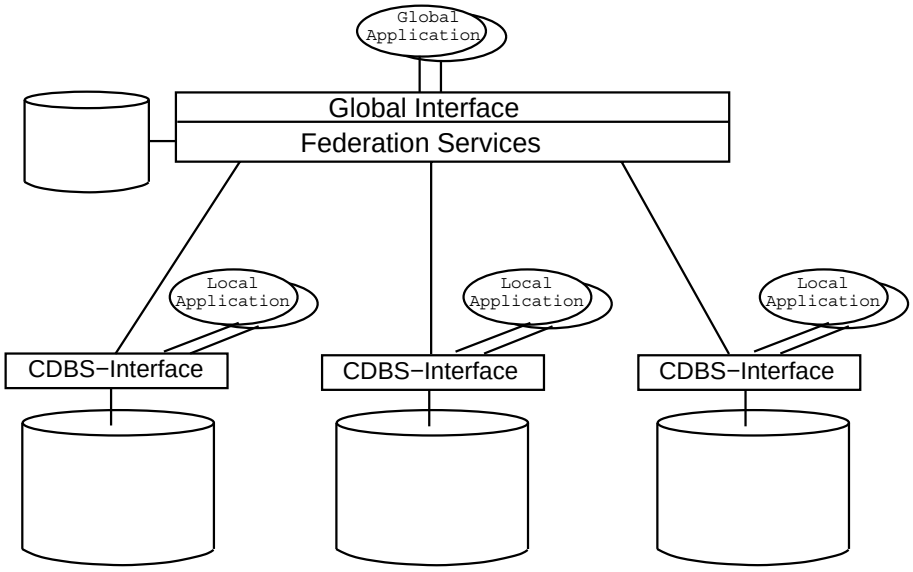

Figure 7: FDBS example out of the Concurrent Engineering area

As prerequisite for object migration, we first consider the schema information: The federated schema is derived from the local schemas (depicted in the CDBS boxes) as follows: The types *Part*, *Material*, *Machine*, *Depot*, and *Costumer* are directly mapped 1:1 (grey type boxes). Type *Design* of DesignDBS is mapped using filtering to the global type *Prod-Tech* (i.e. not all design data is globally visible) and *Product* of ProductionDBS maps 1:1 to this global type which includes technical data of a product (striped type boxes). *Article* of SalesDBS is directly mapped to a global type *Prod-Comm* containing commercial data of a product like its price. Both global types Prod-Tech and Prod-Comm are derived to a global union type *Product* containing all technical and commercial product data. Type *Method* of DesignDBS is invisible in the federated schema (white type box) and a global new type *EnterpriseInfo* is added to the federated schema, not derived from any CDBS. sUUU Now, we demonstrate the use of the various migration dimensions starting with a fixed object kind and migration degree while varying the operation primitive. Afterwards, the other migration dimensions are also varied.

Operation Primitives

We inspect a partial migration (*migration degree*) of an object that was locally created in DesignDB of type Design and which is visible at the global interface as a reduced federated unique object (*object kind*) of global type Prod-Tech. Such a migration is requested in the following situations, each requiring a different operation primitive:

1. The design is released for production, research on the design is terminated. This

requires an *absolute movement* of the design data to the ProductionDBS.

2. The design is released for production, but research on the design continues. New versions of the design should result in new products for a later time. For this, an *independent copy* of the design data is created in the ProductionDBS.
3. A rough design should be produced as a prototype, extensions and modifications to the design should immediately be considered in the prototype. This requires a *replication* of the design data to the ProductionDBS.

A type migration is not necessary here since in the target CDBS (ProductionDBS) a corresponding local type for Prod-Tech already exists, namely Product.

Migration Degree

In the previous paragraph, we have considered a *partial migration* from DesignDBS to ProductionDBS where only globally visible data is transferred. This is ingenious when specific design data, like an attribute 'last_edit_in_design', is not relevant for any local application of the ProductionDBS. When the data is migrated by absolute movement, the invisible data will be lost for the whole FDBS, but it may support automatic deletion of unnecessary data. A *locally complete* migration is necessary if there is at least one local application in the target CDBS (ProductionDBS) having to process some invisible data of the object. Then the object is migrated with all global visible and invisible data. A locally complete migration is absolutely necessary if a local application should be migrated to another CDBS together with its data and has to run with its old data on the new DBS [15].
By *globally complete* migration, all globally visible data of a global object are migrated from all CDBS to a given target. This induces a clustering of an object's data on the target CDBS. For our design object to be migrated from DesignDBS to ProductionDBS globally complete migration works equivalent to partial migration since it was stored in only one CDBS. But we will see different results by varying the object kind in the following.

Object Kind

In the following, we present some object migration examples with different object kinds. For federated and reduced federated objects, we select a common object example.

o1 is a *global invisible* object of type Method in DesignDBS with design specific data for the design methodology. It cannot be migrated to any other CDBS; its data are not relevant in the context of production or sale. The *global new* object *o2* is of type EnterpriseInfo. It is not stored in any CDBS but in the Global-DB of the FDBS. If its data are required some time by a local application of a CDBS, it may be migrated to it. A *reduced federated unique* object *o3* was already presented in the previous paragraphs: an object of type Design. *o4* is a *reduced federated multiple* object resulting from a replication from design data to ProductionDBS. If it is only accessed by global applications some time, it is advisable to migrate the globally visible data by globally complete absolute movement to the Global-DB in order to speed up access performance. A *reduced federated union object o5* is globally created of the global type Product including both technical and commercial product data. The technical data is stored in DesignDBS and ProductionDBS while the commercial data are kept in SalesDBS. An object migration of *o5* may also go to the Global-DB. Here, by globally complete absolute movement the technical data are transferred from DesignDBS or ProductionDBS and the commercial data from SalesDBS. We see a significant difference between partial migration

of the object and globally complete migration. By partial migration we have to choose a source CDBS so that either the technical or the commercial data can be migrated.

2.3 Migration Functionality

After having specified the concepts of object migration in the previous section by the three migration dimensions *Operation Primitive*, *Migration Degree*, and *Object Kind*, we specify concrete migration functionalities for the global FDBS interface here . We provide the users/administrators with a set of operations to invoke object migration. First, we consider *implicit migration operations* which enable object migration by changing an object's class. These operations are already known from existing DBS (e.g. COMIC [9] and COCOON [19]) and FDBS (e.g. O*SQL [12]). While these functions allow a transparent object migration, they restrict on the operation primitives absolute movement / replication and on the migration degree globally complete and, moreover, require specific type mappings. Therefore we add *explicit migration operations*, by which all migration dimensions are directly specified (also configurable), but in most cases do not allow transparent object migration. Both implicit and explicit operations together result in a flexible object migration mechanism. In the following, we define these operations. We show an example syntax, a C++ binding, that is very similar to the C++ data manipulation language OML of the standardization group ODMG [3].

2.3.1 Implicit migration operations

Implicit migration operations change the object's class. They implicitly invoke an object migration for those cases where source and target class are mapped to different CDBS. We offer an operation to 'add' an object to another class. This results in a replication migration in case both classes are mapped to different CDBS. By 'shift', moreover, an object may change into another class corresponding to an absolute movement if the classes map to different CDBS. The operation primitive "independent copy" cannot be realized by class-change-operations because object identity is retained.
The migration dimensions are specified for the implicit migration operations as follows: The migration dimension *Operation Primitive* is fixed by the selected operation (add/shift) and the *Object Kind* is implicitly known by the specified object, e.g. from type information in the object and schema manager of the FDBS. The *Migration Degree* is fixed as globally complete, i.e. all global-visible data of the object stored in arbitrary CDBS is migrated to that/those CDBS where the target class is mapped to.[3]

The following example illustrates object migration in a university FDBS with an EmployeeDBS and a StudentDBS. It shows that university persons can transparently change their status within the university, e.g. from student to employee, without modifying their global identity but changing the location of their data. The implicit migration operations only require the specification of the object and the target class as input. The source class is implicitly known by the object. Using the C++ binding, it looks as follows:

[3] If we extend it to arbitrary Migration Degree, the original semantics of the class-change-operations will change, e.g. a locally complete migration would change/extend the target class.

EXAMPLE:

```
//declaration
Student   *any_student, *another_student;

//absolute movement of a student into class Employee
any_student = select (Student, matr_nr, 1333333);    //object selection
any_student->shift (Employee);                       //object movement

//a student additionally is in class Employee but remains in Student
another_student = select (Student, matr_nr, 1555555);
another_student->add (Employee);
```

Both operations are transparent, i.e. the user has to specify neither source nor target CDBS. But implicit migration operations do not support all tuples of migration dimensions. They require that source and target class are different and map to different CDBS. For example in the CACE FDBS of Section 2.2.4, technical product data cannot migrate from *DesignDBS* to *ProductionDBS* using these operations. Since the local types *Design* of *DesignDBS* and *Product* of *ProductionDBS* are mapped to a single global type *Prod-Tech*, we cannot specify different source and target classes for moving technical product data from *DesignDBS* to *ProductionDBS*. This would have been possible, if the local types *Design* and *Product* were mapped to two separate global types.

2.3.2 Explicit migration operations

By explicit migration operations all combinations of migration dimensions are supported. In general, they represent non-transparent operations, but we will also figure out in this section some special cases which work transparently.
We offer an explicit migration operation for each *Operation Primitive*, i.e. for absolute movement, replication, and independent copy. Each of them requires as input an object to be migrated, the *Migration degree*, as well as source and target CDBS. The specified object, again, implicitly denotes the migration dimension *Object Kind*. For specifying source and target CDBS, the FDBS generates an identifier for each CDBS (similar to [12]).

The following example shows the explicit migration operations for the C++ binding. Identifiers for CDBS and migration degrees are either offered by constants or predefined variables.

EXAMPLE:

```
obj->move (PARTIAL, DB1,DB2);   //absolute movement of object obj from CDBS
                                //DB1 to DB2 with partial migration degree

obj->replicate (LOCAL_COMPLETE, DB2,DB3);    //object replication

obj2 = obj->copy (LOCAL_COMPLETE, DB2,DB4);  //independent copy
```

In general, the migration operations are not transparent because they require location information for the source and target CDBS. But we also allow to configure these information. When migration, for example, is constantly realized from some legacy CDBS to a single

CDBS of new technology, this CDBS can be configured as *fixed target*. If the Global-DB is taken as fixed target, which is in general also of new technology, we eliminate the need for type migration and for the corresponding costly schema evolution steps. By initializing the Global-DB with the federated schema as conceptual/local schema, we do not need type migration for migration degree A' as well as C.
When migration should be done from a single CDBS, e.g. if this CDBS should be eliminated from the FDBS, then we can configure a *fixed source*.
If both source and target CDBS are configured, the explicit migration operations are completely transparent. They do not require any CDBS parameter. This can be realized in the C++ binding by default parameters which are generated during configuration.
For some special combinations of *Object Kind* and *Migration Degree* we also provide transparent access, although only a target CDBS is configured:

- Global-complete migration to a fixed target CDBS for any object kind:
  As source CDBS for globally complete migration all CBDS are taken. The target CDBS is configured for the FDBS.
- Partial or locally complete migration to a fixed CDBS for global new, federated and reduced federated unique objects:
  The source CDBS is unique and implicitly known for these object kinds and the target CDBS is configured.

2.3.3 Extensions for Migrating Object Graphs

While we restrict on migration of single objects in the previous sections, here we extend the functionality to migrate whole object graphs with one operation of the global interface. This may be used to migrate a data structure composed of multiple related objects at once (Fig. 8).

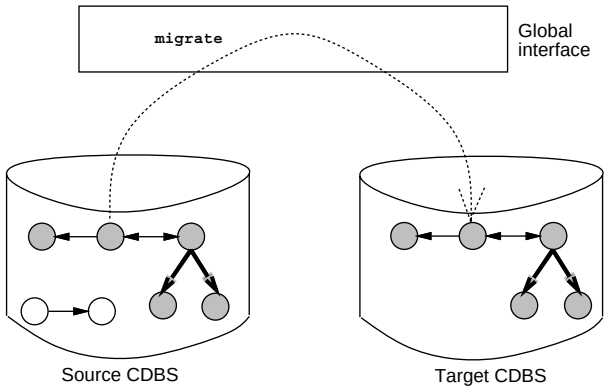

Figure 8: Migration of an object graph with a single global operation

Migration of object graphs can be invoked by the explicit migration operations. Therefore they get an additional parameter, an option for migrating object graphs which is by default OFF. Further on, the operations require a single object as input. If the option for migrating object graphs is set, recursively the specified object, its related objects, their related objects, etc. are migrated automatically. If the canonical data model of the FDBS offers multiple relationship kinds, e.g. COMIC [9] offers general relationships (called relate) and part-of

relationships (contain), we may also restrict the migration of object graphs on one/some relationship kind(s). That means, objects either related by contain-relationships (so called complex objects) or relate-relationships or both are considered. Therefore, the option for migrating object graphs has to specify the relationship kind to be considered. This is indicated in our C++ binding by specific constants: INCL_RELATE for relate-relationships, INCL_CONTAIN for contain-relationships, and INCL_RELATE_CONTAIN for both. Object migration is recursively executed for all objects related to a given object by the specified relationship kind. Migration degree and operation primitive are defined only once and hold for all objects of the object graph.

```
EXAMPLE:

//partial absolute movement of a complex object obj1 with all its components
obj1->move (PARTIAL, DB1, DB2, INCL_CONTAIN);

//partial independent copy of object obj2 together with all related objects,
//i.e. with relate (general) or contain (part-of) relationships
obj2->copy (PARTIAL, DB1, DB3, INCL_RELATE_CONTAIN);
```

Analog to object migration in non-autonomous distributed DBS [5], we may also provide migration of complex objects, i.e. objects with contain-relationships, to a specified deep (n-deep migration). This will require an additional parameter for specifying this recursion level. So, we offer shallow, deep, and n-deep migration. In contrast to [5], FDBS cannot support migration on demand, that is, migration of related objects only upon access to the target DBS. Due to autonomy, a loading on demand could have be realized only for global FDBS applications while local CDBS applications would have to work with incomplete and inconsistent data. In addition to [5], we do not restrict migration to part-of relationships but also consider general relationships to be able to migrate various data structures.

3 Algorithms for Object and Type Migration

In order to specify the required actions for object and type migration, we sketch algorithms in this subsection.

3.1 Object Migration Algorithms

For the three migration operation primitives we present separate algorithms. The various migration degrees and object kinds sometimes require different actions which is mentioned in the corresponding steps of the algorithm. To migrate whole object graphs, i.e. an object together with its related objects, the algorithm is recursively applied to all objects that are related with a given object via a relate and/or contain relationship.

1. Absolute movement
   (a) Determine the relevant data of the global object to be migrated. It depends on the migration degree and the selected option for migrating object graphs:

(a1) For partial migration degree all globally visible data of a given source CDBS, for locally complete migration all globally visible and invisible data of a source CDBS and for globally complete migration all global visible attributes.[4]

(a2) Consider attributes representing relationships, only if the corresponding option for migrating object graphs is set.

If no such data exists, terminate the algorithm.

(b) Check if the type information exists in the target CDBS as prerequisite for the object migration. For locally complete migration an equivalent for the local type has to exist, for globally complete migration and migration variant A' an equivalent for the global type is required and for partial migration the corresponding parts of the global type. If this type information is absent then migrate it (see type migration algorithm).

(c) Read the data determined in step (a) from the CDBS using a read operation of the corresponding CDBS.

(d) Transform the data of step (c) from the local schema of the source CDBS to the local schema of the target CDBS. For partial and globally complete migration degree, transform it from from the source local schema through the federated schema to the target local schema (see Fig. 2). For the locally complete migration degree use a short-cut, i.e. from the component schema of the source CDBS to the component schema of the target CDBS.

(e) Create an object in the target CDBS with the appropriate type (see step (b)) and also assign it to the given global object (i.e. same global object identity).
Write the transformed data of step (d) into this object. Empty attributes are set to Null, e.g. for attributes representing relationships, if no option for object graph migration was chosen.

(f) Delete the corresponding local object(s) of the source CDBS: for globally complete migration degree delete each assigned local object except in the target CDBS, for the other migration degrees delete only those assigned local object(s) that is/are stored in the specified source CDBS, except the source was identical to the target CDBS.

Step (e) was mentioned at the end of the algorithm and not after step (c) to ease FDBS global recovery of these global operations.

2. Replication

(a)-(e) as in 1.

(f) Manage the duplicate data as replicas and guarantee their data consistency (e.g. by techniques of [17]).

3. Independent copy

(a)-(b) as in 1.

(c) Create a new global object and corresponding local object(s) in the target CDBS with the following data: For partial and globally complete migration degree read the selected data of step (a) from the source global object and write this to the new global object (by global read and write operations). For locally complete migration read the selected data of (a) using local operations of the source CDBS and write it to the new global object.

[4]Some code optimization can be done for most object kinds where the actions for the various migration degrees are identical - see interaction between object kind and migration degree, Section 2.2.2.

3.2 Type Migration Algorithms

Type information has to be migrated by replication since it represents meta-information which is further required in the source CDBS and newly in the target CDBS as prerequisite for a migrated object. Type migration induces a schema evolution within the FDBS where some schema levels of the 5 level FDBS architecture are extended.
The type migration algorithms differ for the various migration degrees. They work recursively in the sense that all used types within a type definition (e.g. for establishing a relationship to another type) also have to be migrated if an equivalent type does not exist in the target CDBS. This re-establishes a closure of the target schema.

1. For locally complete migration: migration of the local type
   (a) Determine the type mapping of the local type in the corresponding component schema using the schema mapping information.
   (b) Extend the component schema of the target CDBS with this/these type(s)
   (For conflicts with existing types of this schema, e.g. homonyms, apply schema conflict solution strategies, e.g. [11]. Those strategies may require user assistance.)
   (c) Extend the mapping to the local schema of the target CDBS as well as the local schema itself with this/these type(s) according to the data model mapping.
   (d) Duplicate the mappings of the type(s) between component schema and federated schema from the source CDBS to the target CDBS.
2. For globally complete migration: migration of the global type
   (a) Extend the component schema of the target CDBS with the global type
   (for type conflicts apply schema conflict solution strategies)
   (b) Extend the mapping to the local schema of the target CDBS as well as the local schema itself with this/these type(s) according to the data model mapping.
   (c) Extend the mapping between the federated schema and the component schema of the target CDBS with a 1:1 type mapping between the global type of the federated schema and the new type of the component schema.
   (d) Extend some external schemata and the mapping from federated to external schema if this type should be visible for the users of this external schema.
3. For partial migration: migration of (a part of) the global type
   For migrating the type of a global new, federated unique, or reduced federated unique object the whole global type is migrated, see algorithm 2. The same holds for migration degree A' where globally visible data is migrated into an object of the global type. For all other cases a part of the global type has to be migrated:
   (a) Determine the part of the global type, the property set ρ, that is mapped to local types of the source CDBS.
   (b) Extend the component schema of the target CDBS by a new type with the name of the global type and the attributes/methods of ρ (for type conflicts apply schema conflict solution strategies).
   (c) Extend the mapping to the local schema of the target CDBS as well as the local schema itself with this/these type(s) according to the data model mapping.
   (d) Extend the mapping between the federated schema and the component schema of the target CDBS with a mapping between the global type of the federated schema and the new type of the component schema.
   (e) Extend some external schemata and the mapping from federated to external schema if this type should be visible for the users of these external schemata.

4 Conclusion

This paper presented concepts for migrating objects within an FDBS from one component database system to another. The three migration dimensions considering (1) different object kinds, (2) various migration degrees, and (3) different migration functionalities resulted in a flexible and configurable object migration mechanism. We introduced migration operations and presented a C++ binding which is very similar to that of the upcoming industrial standard of ODMG. The operations allow to (a) implicitly migrate by changing the object's class and (b) explicitly migrate objects from one DBS to another. While the former are transparent, but only support few migration cases, the latter support a wide area of object migration requests, but do not allow a transparent migration in most cases. Together, they offer a flexible object migration functionality. The presented algorithms for object migration also provide mechanisms to migrate whole object graphs, i.e. objects together with related objects, by a single migration operation.

Although existing FDBS approaches do not concentrate on object migration, yet, we figured out application areas that require a flexible object migration concept within FDBS. Such applications request object migration as an FDBS internal service. Moreover, our approach provides assistence in migrating local applications from one DBS to another as well as for reducing the number of DBS within an FDBS.
We already validated the need for object migration by some industrial project cooperations. As an important application, the reduction of the huge number database systems within an enterprise was identified. This is published in a companion paper [15]. Currently, we are developing an FDBS in contact with our industrial project partners.[5] To fulfill their requirements, we will also incorporate object migration capabilities.

[5]This work is granted by the ESPRIT project JESSI COMMON FRAME No. 7364.

References

[1] AGRAWAL, D., ABBADI, E.A. Using data migration for heterogeneous databases. In *Proc. 1st Int'l Workshop on Interoperability in Multidatabase Systems (Kyoto, Japan)*, pages 226–229, 1991.

[2] AHMED, R., ALBERT, J., DU, W., KENT, W., LITWIN, W.A., SHAN, M.C. An overview of Pegasus. In *Proc. 2nd Int'l Workshop on Interoperability in Multidatabase Systems (Vienna)*, pages 273–277, 1993.

[3] CATELL, R.G.G. *The object database standard: ODMG'93*. Morgan Kaufmann Publisher, 1994.

[4] DALE, R. Database migration: keeping a steady course. *Database Programming and Design 3(4)*, pages 30–38, 1990.

[5] DOLLIMORE, J., NASCIMENTO, C., XU, W. Fine grained object migration. In *Proc. Int'l Workshop on Distributed Object Management (Edmonton, Canada)*, pages 181–186, 1992.

[6] ELIASSEN, F., KARLSEN, R. Interoperability and Object Identity. *SIGMOD RECORD 20(4)*, pages 25–29, 1991.

[7] HAERTIG, M., DITTRICH, K.R. An object-oriented integration framework for building heterogeneous database systems. In *Proc. IFIP-DS5 Semantics of Interoperable Database Systems (Lorne, Australia)*, pages 32–61, 1992.

[8] HEIMBIGNER, D., McLEOD, D. A federated architecture for information management. *ACM Trans. Off. Inf Syst. 3(3)*, pages 253–278, 1985.

[9] KACHEL, G., RADEKE, E., HEIJENGA, W. COMIC - A step toward future data models. Cadlab-Report 3/92, 1992.

[10] KENT, W. The breakdown of the information model in multi-database systems. *SIGMOD RECORD 20(4)*, pages 10–15, 1991.

[11] KIM, W., CHOI, I., GALA, S., SCHEEVEL, M. On resolving schematic heterogeneitty in multidatabase systems. *Distributed and Parallel Databases 1*, pages 251–279, 1993.

[12] LITWIN, W. O*SQL: a language for multidatabase interoperability. In *Proc. IFIP-DS5 Semantics of Interoperable Database Systems (Lorne, Australia)*, pages 114–133, 1992.

[13] MEIER, A., DIPPOLD, R. Migration and co-existence of heterogeneous databases; practical solutions for changing into the relational database technology (in german). *Informatik-Spektrum 15(3)*, pages 157–166, 1992.

[14] ÖSZU, M.T., VALDURIEZ, P. *Principles of Distributed Database Systems*. Prentice Hall Publishing, 1991.

[15] RADEKE, E., SCHOLL, M.H. Federation and stepwise reduction of database systems. In *Proc. 1st Int'l Conf. on Applications of Databases (Vadstena, Sweden)*, 1994.

[16] SHETH, A., LARSON, J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys 22(3)*, pages 183–236, 1990.

[17] SRINIDHI, H.N. Management of redundant data in interoperable environments. In *Proc. 2nd Int'l Workshop on Interoperability in Multidatabase Systems (Vienna, Austria)*, pages 236–239, 1993.

[18] TEMPLETON, M., BRILL, D., CHEN, A., DAO, S., LUND, E., Mc GREGOR, R., AND WARD, P. Mermaid: A front-end to distributed heterogeneous databases. In *Proc. IEEE 75*, pages 695–708, 1987.

[19] TRESCH, M. Dynamic evolution in object databases (in german). PhD thesis, University of Ulm, Germany , Feb. 1994.

[20] TRESCH, M., SCHOLL, M.H. Schema transformation processors for federated objectbases. In *Proc. 3rd Int'l Symp. on Datebase Systems for Advanced Applications (Daejon, Korea)* , 1993.

[21] TURINO, J. *Managing concurrent engineering – buying time to market.* Van Nostrand Reinhold Publishing, 1992.

[22] WOELK, D., SHEN, W.-M., HUHNS, M., CANNATA, P. Model driven enterprise information management in Carnot. MCC Technical Report, Nr. Carnot-130-92 , 1992.

Contents