*Apppears in: Proc. Int'l Conf. on Parallel and Distributed Information Systems (PDIS), Austim, Texas, Sept. 1994.*

# Framework for 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 provide a framework incorporating a flexible object migration mechanism into FDBS. It offers different capabilities for "what" and "how" to migrate: (1) different kinds of objects that exist within an FDBS, (2) various migration degrees that determine what data to migrate of an object, and (3) different operation primitives to move, replicate, and copy 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 heterogeneous 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 [17], Carnot [20]) 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 [14] 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. 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 **approach** for object migration in an FDBS. Therefore we con-

sider general FDBS which couple multiple heterogenous database systems, preserve the autonomy of the DBS, and provide a uniform transparent global interface to access data of the various DBS [16]. We investigate the migration of data from one component database systems to another, invoked by the global FDBS interface. It is called "object migration" because the global interface is based on a canonical object-oriented data model in our approach so that data of the multiple heterogeneous DBS are accessed in terms of objects. Beside a base model for object migration in FDBS we come up with a framework which allows to specify different kinds of "what" to migrate from an object and "how" to migrate it.

**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 most 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, 12] 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 [13, 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 base migration concepts from distributed DBS, e.g. the need to migrate the object's type as prerequisite [5]. Due to a different architecture of FDBS which also considers autonomy and heterogeneity, some more differentiated concepts are required. For example, we distinguish different object kinds depending on the visibility of the associated CDBS data.

This paper starts with a base model introducing the preliminary concepts of object migration in FDBS. Then we present a framework for object migration which differentiates different kinds of "what" and "how" to migrate. It considers different object kinds of an FDBS, migration degrees that determine what data to migrate of an object, and some operation primitives specifying whether data is moved, replicated, or copied.

## 2 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. An example for object migration is given in Fig. 1.

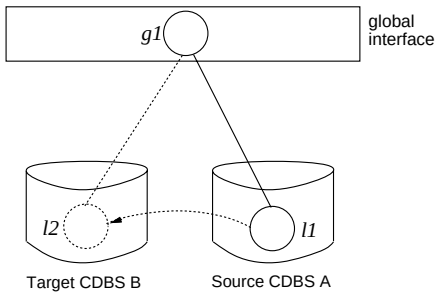


Figure 1: Object migration from CDBS A to B

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*.

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

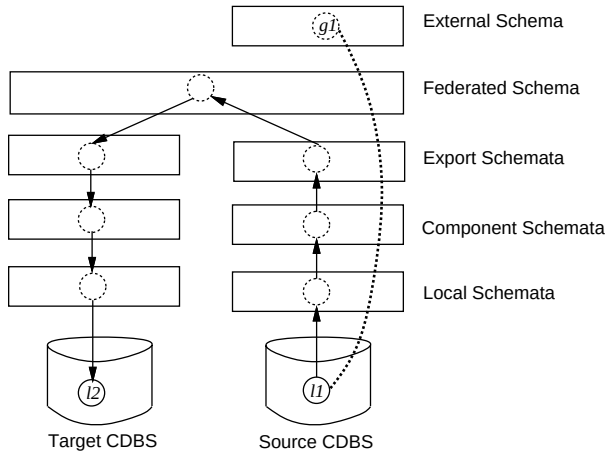


Figure 2: Object migration according to the 5 level FDBS architecture

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 target CDBS as local object(s). At the end, the global object *g1* is (also) associated to the local object *l2*. Therefore subsequent operations on *g1* will (also) have an effect on *l2*.

The scenario of migrating object from one CDBS to another is similar to reading data from some CDBS and writing it to another. 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).

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 [10] which is very similar to the upcoming standard of ODMG [3] both in semantics and syntax of the interface (C++ oriented). The object notion of both models is as follows:

> An object consists of several attributes that express its structure and owns some methods describing the object's behavior. 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:

- *Data attributes* are migrated.
- Attributes representing *relationships* to other objects as well as related objects are not migrated in the object migration base model to the target DBS.
- *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 a corresponding 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 where the local schema of the target CDBS and bottom up also other schema levels are updated (the corresponding component, export, and possibly federated and external schema). This reduces the autonomy of the CDBS. But since we only *extend* schemas, i.e. the schema evolution is capacity preserving according to [9], local applications of the CDBS are further executable without any modification.

Important is that the FDBS supports transparency so that changes in data locality by object migration will be hidden from global applications. Hence, global applications do not have to be recoded if some object is migrated to another CDBS. Also relationships will be preserved at the FDBS global level. When two related global objects as well as their relationship were all mapped to the same CDBS and object migration changes the locality for one of the objects then the relationship becomes an inter-database relationship between objects of different CDBS and is now stored in the internal database of the FDBS (GlobalDB) which stores meta data and global new information. Although the relationship is deleted in the source CDBS by referential integrity rules it will be preserved

in the FDBS to enable compatibility to existing global applications.

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

## 3 Migration Dimensions

Our object migration mechanism considers three orthogonal 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

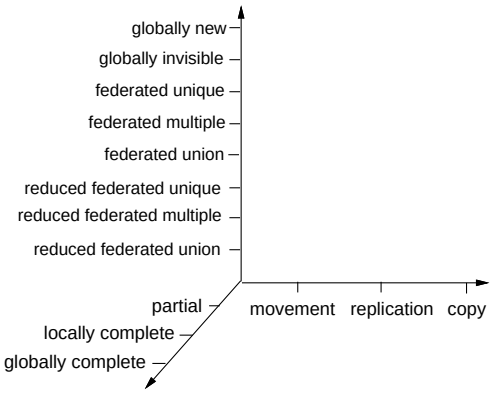


Figure 3: Object migration dimensions

In the following, we first specify these dimensions in more detail and then illustrate them by an example.

### 3.1 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 [11] 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.
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.

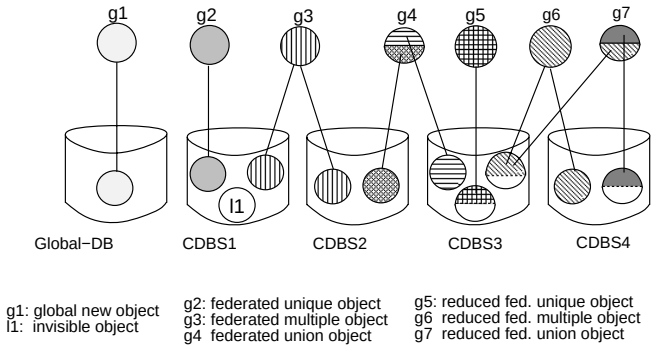


Figure 4: Object kinds in an FDBS

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 [18]. 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.

### 3.2 Migration Degree

The migration degree specifies how much of a global object shall be migrated. We distinguish three migration degrees. In Fig. 5, 6, 7 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' 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).

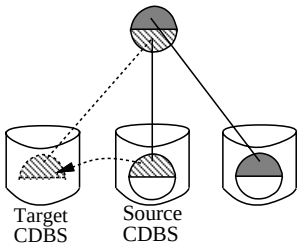


Figure 5: Partial migration degree

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

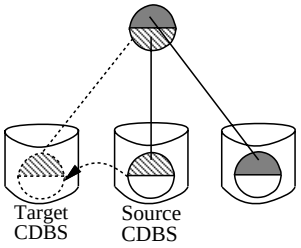


Figure 6: Locally complete migration degree

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

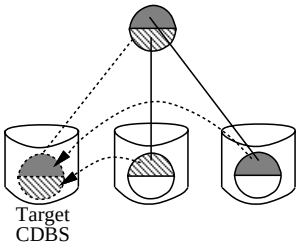


Figure 7: Globally complete migration degree

### 3.3 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 the Figures 8, 9, 10 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.

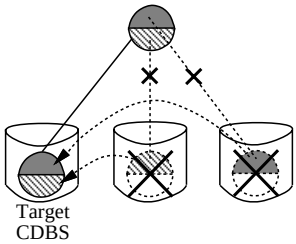


Figure 8: Operation primitive absolute movement

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 [20].

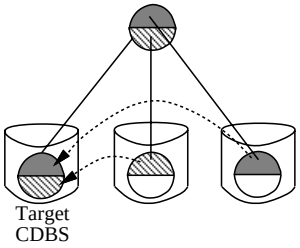


Figure 9: Operation primitive replication

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.

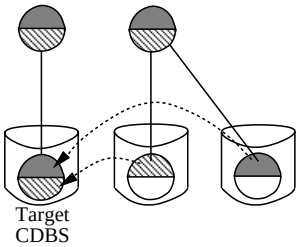


Figure 10: Operation primitive independent copy

If some data of an object has already been stored redundantly in source and target CDBS, e.g. in case of federated multi objects or 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 within an enterprise [14].

### 3.4 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) [19].
The CACE FDBS consists of three component database systems, one for each department of the enterprise. They are: DesignDBS, ProductionDBS and SalesDBS (Fig. 11).

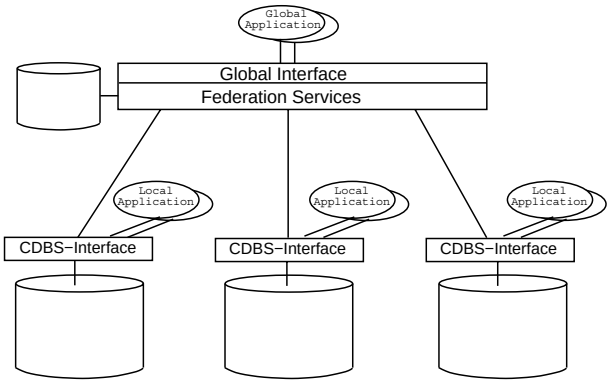


Figure 11: FDBS example out of the Concurrent Engineering area

As prerequisite for object migration, we first consider the schema information: The federated schema is derived from the local schemas as follows (Fig. 12):
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 (dashed type boxes). *Article* of SalesDBS is directly mapped to a global type *Prod-Comm* containing commercial data of a product like its price (striped type box).
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.

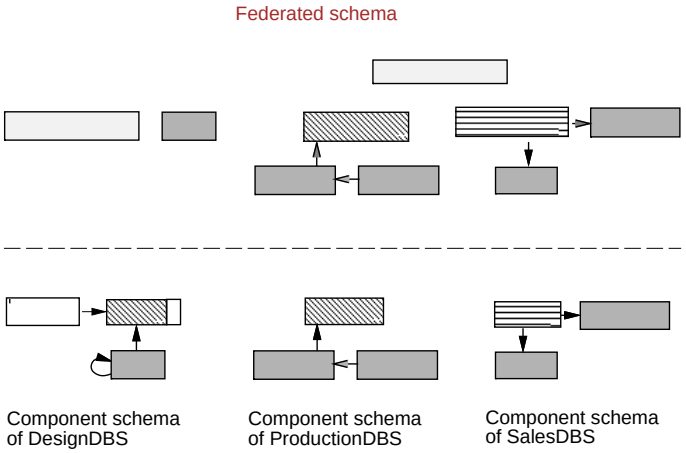


Figure 12: Schema integration for the FDBS example

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 [14].

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.

## 4 Conclusion

This paper presented a framework 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 operation primitives offer flexible object migration facilities. They guarantee compatibility to existing global applications of the FDBS by enabling object identity preservation. Moreover, they allow to control data consistency during the migration phase by the FDBS.

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. It can serve as an FDBS internal service, supports the migration of DBS applications among some database systems, and provides assistance in the reduction of 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 [14].

In order to invoke object migration by the global interface of an FDBS we have specified operations which cover all the migration dimensions of this paper. The

specification of these operations were also submitted but left due to space restrictions from the final paper version. They can be read in [15].
Currently, we are developping an FDBS with a canonical object-oriented data model including migration functionality. In order to validate its industrial relevance the development is realized in contact with some industrial project partners.[1]

The presented framework of this paper restricted on base dimensions for object migration. In order to consider specific application needs, it can be extended by additional dimensions. For example, a dimension "temporality" may limit the migration to a time frame which allows both temporary and persistent object migration. This can be simulated by the presented framework with migration and subsequent local/global deletion operations.

## 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.


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


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

[9] R. HULL. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing 15(3)*, Aug. 1986.

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

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

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

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

[14] 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.

[15] RADEKE, E., SCHOLL, M.H. Object migration in federated database systems. Cadlab Report 3/94, 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] 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.

[18] 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.

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

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