Federation and Stepwise Reduction of Database Systems

Elke Radeke[1] and Marc H. Scholl[2]

[1] Cadlab (Cooperation University of Paderborn & SNI AG),
Bahnhofstr. 32, 33102 Paderborn, Germany, e-mail: elke@cadlab.de
[2] University of Ulm, Faculty of Computer Science,
89069 Ulm, Germany, e-mail: scholl@informatik.uni-ulm.de

Abstract. Many enterprises suffer from the problems of heterogeneous data (base) systems, such as redundancy and lack of control. Existing approaches either try to couple the database systems into a federation (FDBS) or migrate data to a single (often new) database system. This paper proposes an integration of both concepts by incorporating migration facilities into an FDBS. General FDBS concepts couple the multiple database systems and allow several degrees of global *control* over heterogeneity and redundancy. In addition, new migration concepts enable data and applications to move from one component database system to another. This supports the requirement of industrial users to stepwise *reduce* heterogeneity and redundancy in a controlled way. It allows to decrease the number of database systems as much and as fast as the enterprise areas can bear it.

1 Introduction

Today, many data management systems exist for different purposes. Within an enterprise or cooperation different database systems or different releases of a database system are in use. This **heterogeneity** primarily has *technical reasons* since no database system is general enough to be appropriate to all application domains. Moreover, it has *economical reasons* because there are various database systems on the market suitable for the same application domain which differ in service support and price. In general, also *organizational reasons* exist if each department can choose on its own a database system without a global strategy. Huge investments have already been realized for the multiple database systems in the enterprises: many database tools and programs are self-implemented or acquired and gigabytes of data are stored in the databases. But in most enterprises, links between heterogeneous systems are missing. Such a situation induces serious **problems**, as already discussed in the scientific world, but also recognized by industrial organizations. Concerning this problem, we are in contact with two departments of Cadlab's industrial partner Siemens Nixdorf Informationssysteme (SNI). Many organizations and also both industrial project partners requested a reduction of heterogeneity as future global enterprise strategy due to the following problems:

- There are applications that need to access more than one database system. So, application programmers have to use *various database interfaces*, need to know the databases (*location*) of required data, and if the database systems support different data models, also a *conversion* from one model to the other has to be implemented. This increases the code and decreases its clarity. Both of our project partners require such multidatabase applications.
- Some data may be stored *redundantly* in many databases, possibly in *different representations*. Hence, the risk of *data inconsistency* is huge and the risk for wrong decisions increases. Updates on redundant data have to be invoked in multiple database systems by the user. By analyzing the data of one of our project partners, we found geometry information for hardware components both in a PCB layout system and in a part library. Corresponding data of these systems were identified in a self-implemented cross reference list. Updates on hardware components have to be executed on several systems and concurrency conflicts as well as incomplete operation sequences result in inconsistent data.

In order to improve the situation and to fulfill the requirements of industrial users, **our solution** proposes to stepwise *reduce* heterogeneity under the *control* of a federated database system (FDBS). First, existing (and new) database systems are coupled to an FDBS [17] which commonly offers a uniform transparent interface for all database systems and allows to control some kind of redundancy. To enhance performance and decrease (uncontrolled) data redundancy as well as database system heterogeneity, we introduce migration mechanisms for an FDBS. They support an enterprise in minimizing the number of database systems as much and as fast as the areas can bear it. In the long run, it allows to eliminate database systems of antiquated technology (where possible) or to reduce the number of database systems appropriate for the same application domain. To achieve this, their data and applications are migrated (Fig. 1).

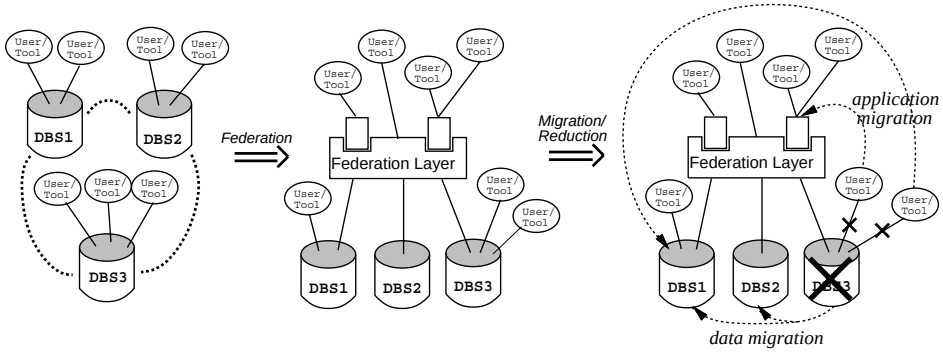

Fig. 1. Stepwise federation and reduction of database systems

In contrast to our approach, some database vendors propose to directly switch from the multiple heterogeneous systems to their single database system. Although a single database system would provide a more performant access to all data than in an FDBS containing multiple DBS, it is neither financially nor organizationally advisable to migrate all existing data and applications to a new

DBS at once, especially in large enterprises. This was also validated by our industrial project partners. Such a migration, in general, lasts several months or even years where the various enterprise areas realize the migration [10]. In this critical phase where data are duplicated to the new database system the enterprise would have no control over these redundant data.
To enable global control when migrating from multiple heterogeneous information systems to a single database system, Drew [4] developed five steps: (1) integrated data access, (2) consistency between redundant data, (3) data migration to a new DBS, (4) application (interface) migration to this DBS, (5) DBS evolution. FDBS were sketched as appropriate solution, but without presenting concrete concepts. Therefore, we close this gap, but regard a more general approach which does not necessarily end in a single new DBS. In many cases and also in one of our industrial projects a single database system will not fit all application domains (see above: technical reason for heterogeneity).
Existing FDBS, e.g. Pegasus [13], Multibase [8], and Carnot [20], all provide a uniform global interface for multiple heterogeneous DBS and support some kinds of redundancy control (detailed comparison below). But no existing FDBS approach elaborates on the reduction of component database systems, which is a typical global enterprise goal.
Gateway database systems, e.g. Oracle, also allow to access data of some different DBS and, in addition, support the importation of foreign data. But they do not control the redundancy between connected database systems. Hence, the risk for data inconsistency is huge.
The **structure of this paper** is as follows: Section 2 presents step one in overcoming database system heterogeneity: The database systems are coupled to an FDBS which *controls* heterogeneity with general FDBS mechanisms. Step two is then realized by a new FDBS concept, object migration, introduced in Section 3. We demonstrate how this can be used to *reduce* heterogeneity and redundancy in an FDBS in a controlled way. The number of database systems can be decreased preserving data and applications. Finally, Section 4 concludes and gives an outlook on future activities.

2 Building a Federated Database System

A federated database system brings the multiple heterogeneous database systems of an enterprise under a global control. It represents an important first step to overcome the heterogeneity and builds the basis of our work. Section 2.1 presents required concepts of an FDBS in form of a global functional model. It is illustrated by specific user requirements that we got from our project partners. While a homogenization layer on top of the database systems hides heterogeneity, specific techniques are required to identify and control data that is redundantly stored in the multiple component database systems (CDBS). They are presented in Section 2.2, based on our prior work. Here, we also show the problems in guaranteeing consistency between redundant data of different CDBS which may result in temporary inconsistencies. Beside the global strategy of many enterprises to reduce heterogeneity, this is a main reason why additional FDBS migration

concepts are required. Before these new concepts are introduced in Section 3, the base concepts of an FDBS are illustrated by a simplified scenario of one of our project partners in Section 2.3.

2.1 Global Functional Model

During the requirement specification with our two project partners, we developed a global functional model for an FDBS (Fig. 2).

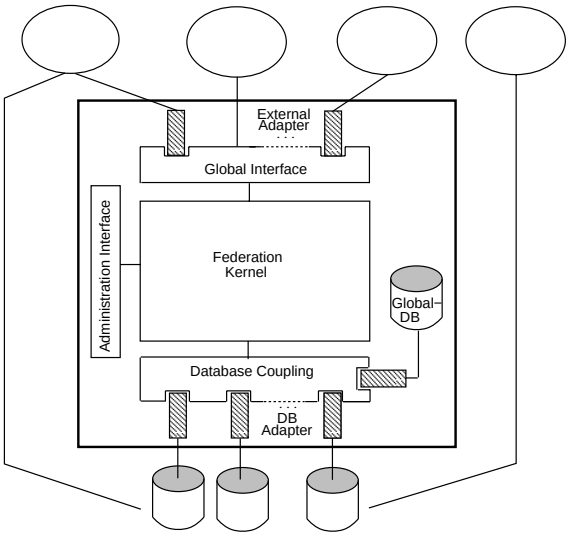

Fig. 2. FDBS architecture

A software layer is built on top of multiple heterogeneous, autonomous, distributed data management systems. Our project partners require to couple hierarchical, relational, and object-oriented database systems as well as file systems. One of them, for example, has 20 different data management systems. While object-oriented database systems are not yet in use at them, all other data management systems already contain a huge amount of data and have multiple applications. Due to lack of resources, these (local) applications should first continue running on their data management systems. This will be achieved by preserving the autonomy of the data management systems. Later on, these applications may be migrated, see Section 3.4.

For new applications accessing data of multiple data management systems, an interface is requested enabling transparent and uniform access to all data. We chose a multi-lingual interface similar to HD-DBMS [2] with a *global interface* and some *external interfaces/adapters*. It was required by our project partners to support their own application interfaces as well as various standard interfaces like STEP (data exchange format), SQL (relational query language), and ODL/OQL/OML (proposed standard of ODMG for object-oriented DBS) which open a wide market of standard software. In Section 3.4, we will see that a multi-lingual interface also eases the migration of applications.

Administration functionality is offered by an *administration interface*. It allows

to start, stop, and initialize an FDBS as well as to to couple/decouple the data management systems to/from the FDBS. A dynamic extension of the FDBS with new data management systems is especially requested by CAx[3] tools of one project partner. CAx tools are often delivered by suppliers with new data libraries at one shot containing new hardware component data.
The coupling software to the component database systems consists of a generic part, the *database coupling* where specifics are realized in some *DBS adapters*. This eases implementation effort, in particular with respect to dynamic extensions with new data management systems.
The *FDBS kernel* handles general required mechanisms like global access control, global transaction management, query/update processing and optimization, data redundancy control (see also Section 2.2), and object migration (see Section 3.1). Meta information as well as global data that do not map to any DBS are stored in an auxiliary database, the *Global-DB*. In Section 3.5 we will see how all existing DBS of an FDBS may be reduced to an integrated DBS in some cases and all data are migrated to the Global-DB.

2.2 Redundancy Control

After the presentation of a functional model for FDBS to globally control heterogeneous database systems, in this section we point out specific techniques to control redundant data. It represents the basis for our migration facilities.
When database systems are coupled to an FDBS, in general they contain many redundant data. Kent [7] called it a mapping from many proxies to one real world entity which require specific object identification techniques [5, 6]. For example, one of our project partners stores geometry information for hardware components redundantly in a layout system and a part library because it is needed by their local applications. In order to reduce the risk for data inconsistency and ease global enterprise management, the redundancy need to be controlled globally within the FDBS.
Redundancy control consists of two steps: (1) identification of redundancy and (2) consistency control of the identified redundancies. Based on our previous work [16, 19], we present two mechanisms to identify redundancy: schema integration and object integration. Afterwards, we mention some techniques for consistency control between redundant data.

Redundancy Identification

Schema Integration:

For hiding heterogeneity and identifying redundancies among schema data, in [19] we developed concrete schema transformation processors. They allow to integrate flexibly the schemata of multiple component database systems (CDBS) and build up the 5 level schema architecture for FDBS [17] (see also Fig. 7). It considers also an incremental evolution of the FDBS, e.g. by extending it with

[3] Computer Aided x where x is a place-holder for Design (CAD), Manufacturing (CAM), Concurrent Engineering (CACE), etc.

new database systems (required by our project partners).
By schema integration, those data *redundancies* can be controlled that result from *globally created* objects. Assume a new object $g1$ is created via the global interfaces of a global type T of a federated/external schema and T was mapped by schema integration to multiple local types $T1, ..., Tn$ ($n > 1$) of some local schemata. Then the FDBS stores this logical global object $g1$ as multiple local objects $l1, ..., ln$ in the corresponding component databases of $T1, ..., Tn$ (Fig. 3). This kind of data redundancy is automatically identified and the mapping between them is managed by the FDBS. By building an FDBS for our project partner that integrates schemata of the layout system and part library such that geometry information of both systems are represented by a single global type, we enable to create new hardware geometry data at one shot in both systems. At present, two separate create-operations have to be invoked and corresponding data is identified by inserting them in a self-implemented cross reference list.

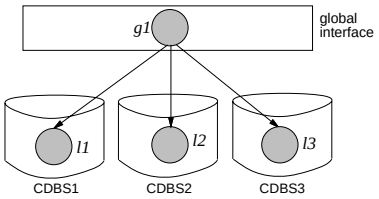

Fig. 3. Object creation for a global type mapped to multiple local types

Object Integration:

Since the database systems of an FDBS were populated independently before the federation, there is already redundant data in the various DBS. In order to supplementary control data *redundancy* that was *created before federation resp. by local applications* in the FDBS, we additionally offer techniques for object integration. They allow to identify existing data of different CDBS as a single entity for the FDBS [7]. For that, we developed a global operation 'same' for the global/external interface [16]. It enables a user/administrator to select some objects via queries and unify them to a single logical global object (Fig. 4).

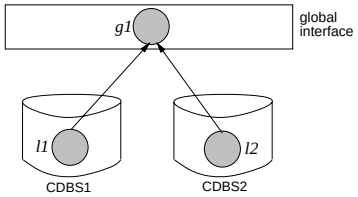

Fig. 4. Identifying two local objects as a single global object

In the FDBS of our project partners, a chance exists to formally specify redundancies among the component systems, e.g. by announcing the self-implemented cross reference list, and let the FDBS control its consistency.

Consistency Control Between Redundant Data

After some objects were identified as same/redundant, global queries will only deliver them as a single global object and the FDBS controls consistency between

their data. This is an important advantage for our project partners. One of them currently handles data consistency of geometry information stored both in the layout system and the part library by hand. By scanning the self-implemented cross reference list from time to time, values of related objects are compared and inconsistencies are corrected by hand. If redundancy is identified in the FDBS, Updates are propagated to the corresponding objects for some degree. This controls consistency and will free our project partners from checking it by hand.
Update propagation is relatively easy for FDBS-global updates but induces some problems for CDBS-local updates. Due to autonomy, local CDBS applications may update objects without effecting the same-objects in the other CDBS. Proposed solutions are to lose the autonomy, using triggers [5], or allowing temporary inconsistency in the FDBS and, for example, specify time frames when data consistency is checked [20]. But some systems of our project partners require autonomy and only a few offer a trigger mechanism (neither IMS nor file system). Thus the FDBS can only support globally controlled, periodic consistency check. That is, data may remain somewhat inconsistent, though identified by the FDBS as redundant. This is an important reason to apply, in a second step, migration concepts (Section 3) to reduce redundant/same objects, after it was brought under global control in the FDBS. Nevertheless, the required global enterprise strategy to reduce heterogeneity is more important.

2.3 Example

Before the migration concepts are introduced, the base concepts of an FDBS are illustrated by a simplified scenario of one of our project partners. It originates from Computer Aided Concurrent Engineering (CACE) [3].
Fig. 5 illustrates a situation in an enterprise containing three database systems for the production life cycle: *ProductionDBS* in the production department with *Machine*, *Material*, and *Product* information; *SalesDBS1* in the first sales department and *SalesDBS2* in the second sales department, both storing data about *Articles*, *Costumers*, and *Depots* of their district.
These three database systems are federated into an FDBS and heterogeneity is hidden by a transparent global interface (object-oriented) and an external interface with SQL syntax for ad-hoc queries. There are several local applications for *ProductionDBS*, *SalesDBS1*, and *SalesDBS2*, but the enterprise also developed a global application that browses all enterprise information. This global application accesses data from all three database systems through the global interface in a uniform and transparent way.
Redundancy among the database systems is identified by schema and object integration so that the FDBS may control consistency among them to some degree. By *schema integration* we combine the local types *Product* of *ProductionDBS* and *Article* of *SalesDBS1* and *SalesDBS2* to a single global type, named *Product* while all other local types are directly mapped 1:1 to a global type in the federated schema. After schema integration, product data can be created at one shot in all three databases systems. When a global application creates an object *g1* of a global type *Product* this is automatically stored as three local objects, *l1* in *ProductionDBS*, *l2* in *SalesDBS1*, and *l3* in *SalesDBS2* (Fig. 3).

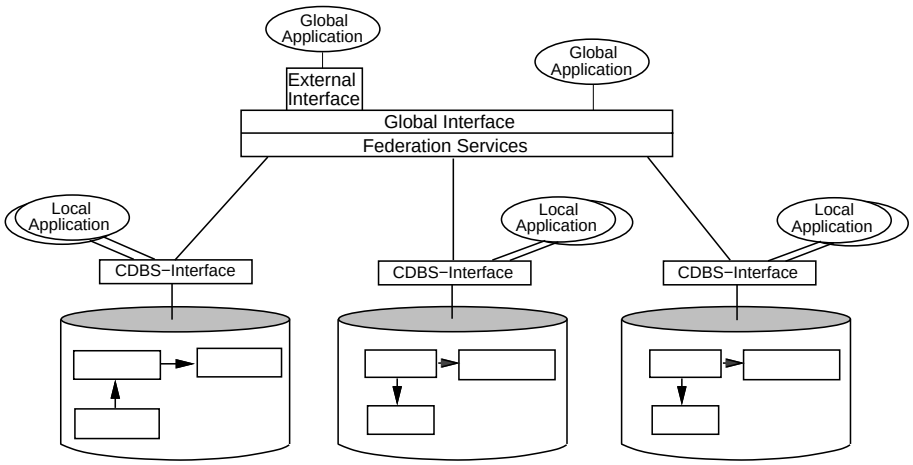

Fig. 5. FDBS for Computer Aided Concurrent Engineering

Moreover, object integration allows to supplementary identify redundant data that was already stored in the database systems before federation or during federation by some local applications. Assume that *ProductionDBS* and *SalesDBS1* both contain a product "SGI-screen" before federation. To let the FDBS know that these two local objects represent a single entity for the FDBS, they are identified as same by object integration (Fig. 4).

While in this section we showed how some concrete database systems get under the global control of an FDBS, in the following we extend the example by demonstrating the reduction of heterogeneity/redundancy.

3 Migration within an FDBS

In this section, we extend the FDBS with new capabilities to migrate objects from arbitrary CDBS to another one. We first introduce base concepts for migrating objects within an FDBS and show how data that was already identified as redundant can be reduced to less local objects using object migration. Then we point out the use of object migration to minimize the number of CDBS. This represents the important second step in overcoming heterogeneity and easing enterprise management. It supports the global strategy of many organizations including our project partners to reduce heterogeneity in a controlled way.

While we restrict on base concepts for object migration in this paper, we present more details in a companion paper [12], e.g. underlying algorithms or an analysis of migration for other objects than those mapped directly 1:m to local objects.

3.1 Base Model for Object Migration in FDBS

By object migration, data is transferred from one component database system of the FDBS to another. Users can initiate object migration by an operation of the FDBS global (and external) interface. Therefore they have to specify the

(global) object and, in general, source as well as target CDBS.[4]
We extend the CACE FDBS example of Section 2.3 in order to illustrate this:

EXAMPLE 2: Assume a new product is released for sale in district 1 and, hence, migrates from ProductionDBS to SalesDBS1 (Fig. 6).

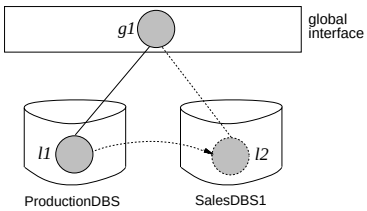

Fig. 6. Object migration from ProductionDBS to SalesDBS1

Here, global object *g1* that was stored in ProductionDBS as local object *l1* shall be migrated through the global interface to SalesDBS1. We also see that object migration requires additional capabilities for object identification. The assignment of global to local objects must be updatable dynamically. In the example of Fig. 6 global object *g1* was first assigned to the local object *l1* and after object migration also/instead to local object *l2*. ♢

What data migrates of an 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 fulfilling the requirements for an FDBS canonical data model [15]. An upcoming standard for an object-oriented data model is fixed in [14] with the following object notion:

An object consists of several attributes that express its structure (data attributes and relationships) and owns some methods describing the object's behavior. To formally define the attributes and methods of objects, types are used. Generally, an object is migrated with the following data to a target CDBS:

- All *data attributes* are migrated.
- Attributes representing *relationships* and related objects are not migrated. When two related global objects and 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 interdatabase relationship between objects of different CDBS and is now stored in the Global-DB. Although the relationship may be deleted in the source CDBS by DBS referencial integrity rules it will be preserved in the FDBS to enable compatibility to existing global applications. In [12] we also consider the migration of object graphs, i.e. objects with related objects.
- *Methods* are only supported by a few DBS, e.g. object-oriented DBS. So 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 (s.b.).

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

– The *type* of an object has to be migrated as a prerequisite if there does not exist an equivalent type in the target CDBS. Two local types of different CDBS are equivalent, if they were mapped during schema integration to the same global type.
  A type migration, in general, induces schema evolution within the FDBS. This decreases the autonomy of the FDBS. But since schema evolution is reduced to schema extension, local applications of the CDBS are further on executable without any modification. There are also some special cases that do not need any type migration at all [12].

When an object is transferred from one CDBS to another, data has to be transformed according to the FDBS architecture. In Fig. 7 we see how an object migration is processed using the 5 level FDBS schema architecture of [17].

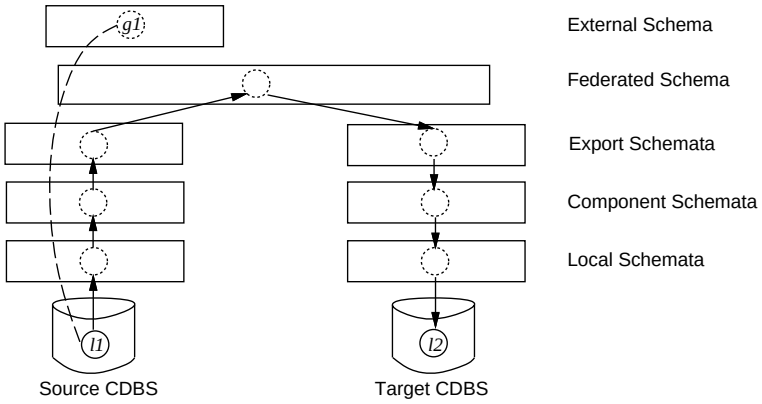

Fig. 7. Object migration according to the 5 level FDBS architecture

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).

3.2 Base Operations for Object Migration

We offer three kinds of operation primitives for object migration (Fig. 8):

1. *Absolute movement*: Objects are transferred into the target CDBS as shown in the base model and are deleted in the source CDBS. Object identity at the global interface remains the same. Shared information, e.g. type information, cannot be migrated by absolute movement.
2. *Replication*: Objects are transferred to the target CDBS but not deleted in the source CDBS. The duplicates represent a single global object and the FDBS may guarantee data consistency between them (see also Section 2.2).

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

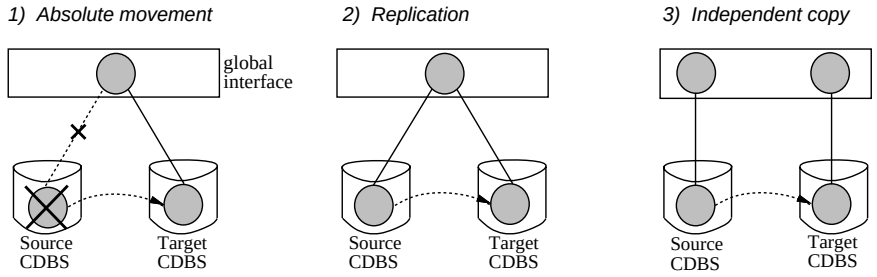

Fig. 8. Migration functionality: (1) absolute movement, (2) replication, (3) independent copy

These migration functionalities are offered at the FDBS global interface and possibly at some external interfaces by (1) explicit migration operations 'move', 'replicate', and 'copy' and the first two also by (2) operations for class change, 'shift' and 'add'.

Explicit Migration Operations

The explicit migration operations, in general, require as input the object to be migrated as well as source and target CDBS. The FDBS generates an identifier for each CDBS to enable the specification of the CDBS (similar to [9]).

EXAMPLE 3: Within the CACE FDBS, single objects can be moved, replicated, and copied across component database systems as follows using a C++ global interface where identifiers for the CDBS are offered by predefined variables:

```
obj->move (SalesDBS1,SalesDBS2);           //absolute movement of 'obj'
                                           //from SalesDBS1 to SalesDBS2
obj->replicate (ProductionDBS,SalesDBS1);  //object replication
obj2 = obj->copy (ProductionDBS,SalesDBS2);//independent copy
```
♢

These migration operations are, in general, not transparent because they require location information for the source and target CDBS. But we allow to configure some parts. For example, the administrator may fix a specific CDBS resp. the Global-DB as target for object migration through the administrator interface. This is ingenious when migration is constantly realized from some legacy CDBS to a single CDBS of new technology. If we may also configure the source CDBS in an FDBS, all migration operations are quasi transparent. That means no CDBS has to be specified in any migration operation call. In [12] we identified further cases of transparent explicit migration operations.

Implicit Migration Operations

Implicit migration operations provide another kind of transparent migration. It can be induced by modifying the object's class. We offer a global 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.

EXAMPLE 4: In a university FDBS with two database systems *StudentDBS* and *EmployeeDBS*, we may arbitrarily change persons from a student to an employee at the end of study (absolute movement), or replicate his data from StudentDBS to EmployeeDBS if he becomes a student assistant and local employee applications require the data. All operations retain the global identity of the person so that global university applications do not have to be modified.

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

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

♢

The implicit operations are transparent because the user has to specify neither source nor target CDBS. But in contrast to the explicit migration operations, it supports fewer cases of object migration. The user has to specify two different classes as source and target and in addition both classes have to be mapped to different CDBS. For the CACE FDBS example, product data cannot migrate from *ProductionDBS* to *SalesDBS1* or *SalesDBS2* using implicit migration operations. The reason is that only a single global type exists for product data though mapped to three local types of different CDBS (*Product* of *ProductionDBS* and *Article* of *SalesDBS1* and *SalesDBS2*). This would have been possible, if these local types were mapped to different global types.
Class change in FDBS was also considered in O*SQL [9] and COCOON [18] and build the starting point for development of our object migration mechanism.

Migration of Object Classes and Databases

Beside migrating single objects, we also provide mechanisms to migrate whole object classes[5] or databases with a single operation. For this, we provide the explicit migration operations also for classes and databases. This corresponds to an iterative execution of object migration for all objects of a given class or database. Although the operations need not be realized as atomic operations because it may imply migration of some thousand objects, it makes the use of object migration easier and more convenient, in particular the reduction of DBS.

EXAMPLE 5: We may replicate all product data of *ProductionDBS* to *SalesDBS2* at once or completely move all data of *SalesDBS1* to *SalesDBS2* with the following operations using a C++ binding where classes and databases are specified by predefined variables:

[5] A *class* comprises some objects of a given type.

```
Product->replicate (ProductionDBS, SalesDBS1);
SalesDBS1->move (SalesDBS2);
```
♢

3.3 Reduction of Redundant Data

A special case of object migration exists for those objects that were stored redundantly in the source and target CDBS and the redundancy was already identified within the FDBS by schema or object integration (Section 2.2). This case can be used to decrease redundancy in the FDBS and may become necessary during the reduction of DBS.

To analyze how migration works for objects that were identified as redundant and stored in source and target CDBS, we regard the three migration operations:

1. Absolute movement: The redundant data of the object only has to be removed from the source CDBS, but, in general, no transfer of redundant data has to be realized.[6] This achieves a reduction of redundant data where the number of CDBS the data is stored in decrements for one.
2. Replication: No transfer of redundant data has to be realized.
3. Independent copy: The copy function operates as usual. A global new object is created and the data is duplicated in the target CDBS. Now, the redundant data will be stored twice in the target CDBS, but since it is an independent copy with different global identity both may change differently.

EXAMPLE 6: Assume a global object $g1$ of type Product is stored as three local objects $l1$ in *ProductionDBS*, $l2$ in *SalesDBS1*, and $l3$ in *SalesDBS2*. They were identified as redundant and consistency between their data was controlled as far as possible from the FDBS. If a global application requests to replicate $g1$ from *SalesDBS1* to *SalesDBS2* then no action has to be invoked because such a replica already exists. If it should be migrated by absolute movement, e.g. when both database systems are reduced to a single one, then the product data is deleted in the *SalesDBS1*. Now, $g1$ is only stored redundantly in two CDBS (instead of three) and the risk for temporary data inconsistency decreases. ♢

Sometimes, not all data of an object is redundant in source and target CDBS but only a subset. Then, non-redundant data of the source-CDBS are migrated as usual. In [12], we differentiate some object kinds dependent on how much data is redundant and where it is stored and show object migration for these kinds. In this section, however, we focus on those objects that were already managed by the FDBS as completely redundant/same. We have seen that they are automatically reduced during object migration. This decreases redundancy and facilitates DBS reduction.

[6] Exceptions may be required if redundant data were not consistent in source and target at the time of object migration. If the data from source CDBS was chosen as correct during conflict solution, however, a migration of redundant data is necessary.

3.4 How Object Migration Supports Application Migration

Increasing requirements of applications to a database system can make it necessary to switch application systems partly or completely to a new database system. Meier and Dippold [10], for example, describe a change of a banking system from hierarchical to relational database technology. Existing literature, e.g. [1, 4, 11], focus on application between isolated database systems. But also in FDBS, application migration is required if existing local applications require to change to newer database technology. We distinguish the following two cases of application migration (Fig. 9) which sometimes require a migration of their application data (object migration):

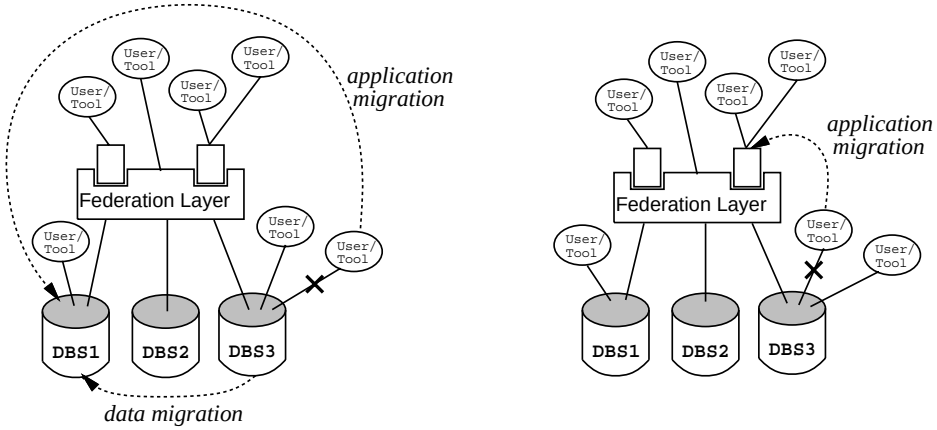

Fig. 9. Application migration (1) to a CDBS interface and (2) to an FDBS interface

1. An application may become a local application of another CDBS. For this, either the application has to be adapted to the new DBS interface or its application interface is previously realized on top of this CDBS. To enable its execution after migration, all its data has to be migrated to the target CDBS, too.
   EXAMPLE 7: In our CACE FDBS we may migrate/replicate a sales application primarily acquired for *SalesDBS1* to *SalesDBS2* in order to locally benefit in both database systems from it. Since both database system are relational and offer the same SQL interface, the application can directly be used in *SalesDBS2* without any recoding after its data was migrated through the FDBS global interface. ♢
2. An application may also get a global application, since FDBS and their global interface mostly underly new database technology. For this, the application is either adapted to the global resp. an existing external interface of the FDBS or the application interface is realized as new external interface and the application remains unchanged. The application data may remain in its source CDBS. Then the migrated application accesses its data through the 5 schema levels of the FDBS (see Section 2.2).
   EXAMPLE 8: If a graphical data monitor of *ProductionDBS* is well known and accepted in the enterprise and also suited as an FDBS monitor, this

local application can be migrated/replicated from the *ProductionDBS* to the global/external interface of the FDBS. In case the application interface is different from the global and any existing external interface, we have to either recode the application to one of these interfaces or realize the application interface as new external interface in the FDBS. ♢

More performant access may often be achieved for an application that becomes a global application if its data are migrated into the Global-DB due to easier mappings.

An easy application migration can be realized when the target CDBS has a similar data model and functionality than its source. If both offer exactly the same interface, the application is directly executable after its data were migrated. The possibility for same interfaces increases if standard interfaces are provided, e.g. SQL of ANSI/ISO for the relational data model.

Application migration and object migration may in particular serve to reduce the number of CDBS in an FDBS. While the general FDBS capabilities presented in Section 2, e.g. schema integration and object integration, allow to control heterogeneity and redundancy in some cases, object and application migration can be used to stepwise reduce heterogeneity and redundancy. This is explained in more detail in the following section.

3.5 Reduction of Component Database Systems

In many enterprises or cooperations there is a multitude of database systems, some ten or hundreds. Such a situation also exists at our project partners. By coupling database systems to an FDBS, heterogeneity is hidden while preserving their autonomy. But for the long term, a reduction of this big number of database systems is desirable. This was in particular formulated as global enterprise strategy at our project partners. A reduction of database systems will result in the following advantages for an enterprise:

- Less effort for database administration if the number of databases decreases.
- In general, there is less redundant data in the FDBS. If redundancy was already identified (by schema/object integration), the effort for consistency control will decrease and the remaining temporary inconsistencies are reduced. For redundant data not yet identified as same, the risk for general data inconsistency will decrease.

In general, no database system can be eliminated together with all its data and applications because some of it is still required somehow. In order to reduce the number of database systems in an FDBS without eliminating relevant data or applications, we recommend migration techniques. Before eliminating a database system from an FDBS, all still required applications are migrated from this CDBS to some arbitrary CDBS or the FDBS interfaces (see previous section) and all corresponding data is migrated by our object migration mechanism to the other CDBS resp. the Global-DB (Fig. 10).

Those data that were already stored redundantly in source and target CDBS and where redundancy was controlled by the FDBS are reduced during object migration. It will not be duplicated in the target (see Section 3.3).

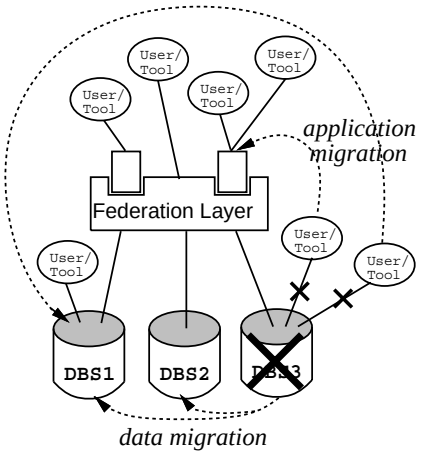

Fig. 10. Database reduction in an FDBS by object and application migration

DBS candidates to be eliminated are those with obsolete database technology that do not fulfill the requirements of today's enterprises [10]. Moreover, multiple database systems supporting the same data model and offering similar functionality may be reduced to a single database system. The latter case particularly eases application migration (see Section 3.4).
EXAMPLE 9: When the two districts corresponding to *SalesDBS1* and *SalesDBS2* are composed together to a single enterprise district, we may also reduce both database systems to a single one. This is quite easy in our example since both DBS are relational with an ISO-SQL interface. After all data of *SalesDBS1* is migrated by absolute movement for databases to *SalesDBS2* or vice versa, the applications are directly executable on the target CDBS. ♢

Our project partners emphasized that not all database systems can be reduced at once. Although the reduction of heterogeneity is the global enterprise strategy, not all enterprise areas may fulfill this goal immediately. Instead a stepwise approach is requested by which the various enterprise areas may reduce heterogeneity and redundancy as much and fast as they bear it. So, global enterprise heterogeneity reduces step-by-step.

Migration Paths

In the following, we discuss by three migration paths in which order data and applications may be migrated to reduce the number of database systems in an FDBS. We show also how our object migration functionality is used for this. Exemplary, we took the Global-DB as migration target.

Path 1 : First, all data of the source CDBS is replicated into Global-DB using the 'replicate'-operation for databases. Afterwards applications of the CDBS may stepwise be migrated by either adapting them to an external/global interface or realizing the application interface as a new external interface.
Advantage: During the migration of the applications (in general some months or years) data consistency may be controlled by the FDBS.

Disadvantage: High expense to manage a completely replicated database. In case of all data not being required any longer, e.g. because some applications will be eliminated, more data than necessary were migrated.

Path 2 : Applications are first migrated to global/external interfaces and data access is realized through the 5 level schema architecture during this time. After all applications were migrated from a CDBS, all the data is migrated via 'move'-operation for databases into the Global-DB.
Advantages: (1) consistent data during the migration of the applications; (2) no data replication and thus no additional effort for replication management.
Disadvantage: Migrated applications run with lower performance as long as data are not migrated to the Global-DB. A heuristic that migrates performance-critical applications at the end weakens this disadvantage.

Path 3 : As a mix of 1. and 2., an iterative process migrates stepwise an application or application set with selected objects to the Global-DB. First, we analyze the schema of the application (set) and then migrate all objects that correspond to the schema information via 'replicate'-operation for classes. Afterwards the application resp. application set is migrated. Thus, stepwise more and more objects are migrated to the Global-DB and managed as replicas by the FDBS. When all applications using a specific type/class of the CDBS schema are already migrated, all objects of this type/class can be eliminated in the source CDBS. To further minimize the number of current replicas, a heuristic based on connectivity components identifies application sets that possess many types together but have less in common with other applications. These application sets should be migrated in connection.

In some cases, by using the technique of database reduction, we may reduce from a multitude of heterogeneous distributed autonomous database systems to a *single integrated database system*. For example, if all CDBS are eliminated, their data is migrated to the Global-DB, and still required applications are adapted to the global or an external interface of the FDBS (Fig. 11).

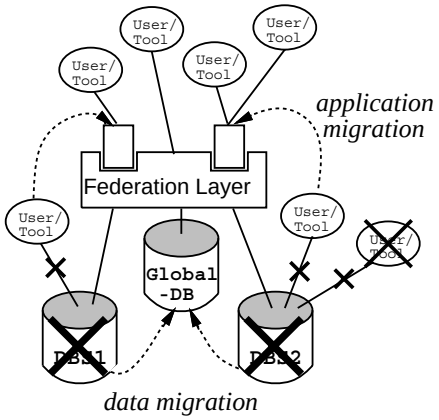

Fig. 11. Database reduction in an FDBS resulting to a single integrated DBS

This new database system still includes federation services. Later on new database systems, e.g. of future database technology, may be coupled together with these data and applications. So, this approach tries to eliminate current heterogeneity in a smooth way but does not represent an isolated system for the future.
In contrast to [4], our projects with two industrial organizations showed that not all enterprises or cooperations will go for only a single database system because a single database system may not fit all their application domains.

4 Conclusion

This paper presented an approach of how enterprises may stepwise overcome the problems of database system heterogeneity. In particular it supports the global goal of our industrial project partners to reduce heterogeneity in a controlled way. The essential idea is the tight integration of migration facilities into an FDBS. General FDBS concepts first *control* heterogeneity and redundancy while the migration facilities allow to *reduce* them under control of the FDBS step by step. We presented object migration concepts for an FDBS which allow to move data from one component database system to another without coercion to change global identity (global data independence). Redundant data is not duplicated in the target CDBS but is automatically reduced (redundancy reduction). We showed how object migration supports enterprises in decreasing its number of database systems while retaining still required data and applications of eliminated DBS. Each enterprise area can choose as much and fast as it bears some database systems to be eliminated. So, the global enterprise goal of heterogeneity minimization can be achieved step by step.
In general, building an FDBS is not as expensive as heterogeneity problems cost an enterprise. Federation services do not represent an enterprise-specific software, but contain many generic parts which can be implemented for several enterprises in common. At Cadlab, we are currently developing federation services containing a generic kernel and plan to realize with our industrial project partners some customers specific FDBS.[7] In order to fulfill our partners' requirements, we will also implement object migration capabilities in our prototypes. While this paper derived from users' needs rough concepts for object migration in FDBS and showed the benefits, more precise concepts as well as realization aspects are elaborated in a companion paper [12]. To further support database reduction, more work is needed to ease the adaptation of applications to new database interfaces. Some approaches already exist in this field [10] which consider application migration between isolated DBS.

References

1. M.L. BRODIE. The promise of distributed computing and the challenges of legacy systems. In *10th British National Conf. on Databases, Advanced Database Systems (Aberdeen)*, pages 1–28, 1992.

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

2. CARDENAS, A. Heterogeneous distributed database management: The HD-DBMS. *Proc. IEEE 75*, pages 555–600, 1987.
3. CHAPPELL, C., STEVENSON, C. *Concurrent Engineering: The Market Opportunity.* OVUM, 1992.
4. P. DREW. On database technology for information system migration and evolution. In *Proc. Workshop on Interoperability of Database Systems and Database Applications (Fribourg, Switzerland)*, pages 121–130, 1993.
5. ELIASSEN, F., KARLSEN, R. Interoperability and Object Identity. *SIGMOD RECORD 20(4)*, pages 25–29, 1991.
6. 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)*, pages 32–61, 1992.
7. KENT, W. The breakdown of the information model in multi-database systems. *SIGMOD RECORD 20(4)*, pages 10–15, 1991.
8. LANDERS, T., AND ROSENBERG, R. An overview od Multibase. In *Proc. 2nd Int'l Conf. on Distributed Databases* , pages 153–184, 1982.
9. LITWIN, W. O*SQL: a language for multidatabase interoperability. In *Proc. IFIP-DS5 Semantics of Interoperable Database Systems (Lorne)*, pages 114–133, 1992.
10. 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.
11. NETZE, J., SEELOS, H.-J. Scenes and strategies of data migration (in german). *Wirtschaftsinformatik 35 (4/93)*, pages 320–324, 1993.
12. RADEKE, E., SCHOLL, M.H. Object migration in federated database systems. Cadlab Report 3/94, 1994.
13. RAFII, A., AHMED, R., DESMEDT, P., DU, W., KENT, W., KETABCHI, M.A., LITWIN, W.A., SHAN, M.C. Multidatabase management in Pegasus. In *Proc. 1st Int'l Workshop on Interoperability in Multidatabase Systems (Kyoto). IEEE Comp. Soc. Press*, pages 166–173, 1991.
14. CATELL R.G.G., editor. *The object database standard: ODMG'93.* Morgan Kaufmann Publisher, 1994.
15. SALTOR F., CASTELLANOS M., GARCIA-SOLACO M. Suitability of data models as canonical models for federated databases. *SIGMOD RECORD 20(4)*, pages 44–48, 1991.
16. SCHOLL, M.H., SCHEK, H.J., TRESCH, M. Object algebra and views for multi-objectbases. In *Proc. Int'l Workshop on Distributed Object Management (Edmonton)*, pages 352–372, 1992.
17. SHETH, A., LARSON, J. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys 22(3)*, pages 183–236, 1990.
18. TRESCH, M. Dynamic evolution in object databases (in german). PhD thesis, University of Ulm, Germany , Feb. 1994.
19. 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.
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.

This article was processed using the LaTeX macro package with LLNCS style