Database Collaboration Instead of Integration

Gunar Fiedler **Thomas Raak** **Bernhard Thalheim**

Kiel University
Computer Science and Applied Mathematics Institute
Olshausenstr. 40, 24098 Kiel, Germany
Email: {fiedler, traak,thalheim}@is.informatik.uni-kiel.de

Abstract

Database integration is currently solved only for the case of simple structures. Semantics is mainly neglected. It is known but often neglected that database integration cannot be automated. System integration is far more difficult. Both integrations can only be performed if a number of assumptions can be made for the integrated system. Instead of integrating systems entirely cooperation or collaboration of systems can be developed and used. We propose in this paper the extension of the view cooperation approach to database collaboration.

Keywords: data integration, data warehouse, view cooperation, database collaboration

1 Database Integration

Nowadays, a large number of very different information systems are used in parallel. These systems coexist in several environments and may be understood as a collection of distributed, redundant, partial, and partially autonomous information systems. Each system contains a specific set of data.

Database integration is an old (Spaccapietra & Parent 1989), a never really solved (Lee & Ling 1995, Lee & Ling 1997) and a very difficult problem. The difficulty is caused by

the ***heterogeneity*** of data both at the intensional and extensional level,

limitations to access the source data,

the decision which data should be ***materialized*** and which should be left to local databases,

data ***extraction, cleansing and reconciliation*** within the database set,

strategies for ***data modification*** processing,

strategies for ***quality management*** of querying, especially statements on whether the data in the query answer is complete and sound,

automatic transformation of ***queries*** posted to the database set, and

expressiveness of ***modeling languages*** aiming at representing the local databases and the integrated databases.

A main problem to be solved in designing such integrated systems lies in *information integration*, i.e., the activity by which different input information sources are merged into a global system describing the whole information set available for query and functionality purposes. Abstraction amounts to clustering types belonging to the schema (Beeri & Milo 1999) into homogeneous subsets and producing an abstracted schema obtained by substituting each subset S with a single abstract type T^S.

Already structural integration (e.g., (Christine Parent 1998, Bernstein & Rahm 2001, Abiteboul, Cluet & Milo 1997, McCabe, Chowdhury, Grossman & Frieder 1999, Conrad, Saake & Sattler 1999)) may become a nightmare. The designer has to clearly understand the semantics of involved database types. In such system re-engineering problems, the design emphasis is on integration of pre-existing information components, where a key problem is that of deriving associations holding among types in the pre-existing schemas. Most of research has been carried out to solve the problem of detection and treatment of *interscheme properties* that relate types belonging to different schemas. The integration of structures and functions (Theo Härder 2002) is far more difficult.

Structural integration problems (Thalheim 2000) such as *structural mismatches* (key differences, abstraction grain, attribute domain, temporal basis, missing parts) *semantic mismatches* (scope difference, value semantics, domain semantics), *operational mismatches*, and *application domain mismatches* may be either treated by full integration, integration by merging, or integration by generalization. In the literature, many *"manual" methods* (Thalheim 2000) for deriving interschema properties have been proposed. A major limit of manual methods relies in the difficulty of carrying them out to large applications since, in such contexts, it is needed to face integration problems often involving hundreds of types.

Since an automatic support to integration of systems cannot be developed, *semi-automatic methods and tools* have been developed or proposed to face the difficulties. Systems such as Autoplex, automatch, Clio, COMA, Cupid, Delta, DIKE, EJX, GLUE, LSD, MOMIS, ARTEMIS, SemInt, SKAT, Similarity Flooding (SF), and TranScm mainly have emerged from specific applications. A very few approaches (Clio, COMA, Cupid, and SF) try to address the schema matching problem in a generic way. All of them are, however, only treating simple structural concepts and none of them treats functionality.

Integration of several information resources requires, however, knowledge describing their contents in a logical formalism and using the same vocabulary. This provides shared access to multiple information sources and preserves at the same time the autonomy of each source. This approach is known as the *mediator* approach (Wiederhold 1995, Cluet,

Delobel, Siméon & Smaga 1998, Papakonstantinou & Velikhov 1999, Ludäscher & Gupta 1999). Mediators play the role of an interface between the user and the sources and between the sources giving the illusion of querying a central and homogeneous system.

Interoperability of database and information systems has been a major research area (e.g., (Sadri, Subramanian & Lakshmanan 1996)) in the last decade. After solving problems with scalability and parallelism rich semantic models have been used for developing federated and extensible database systems.

Potential database integration depends on early modeling assumptions and is thus dependent on a number of *implicit assumptions* made during the development process:

- Database development is ruled by a number of *implicit points of view*. Depending on what has been the main target and scope, basic structures and domains are chosen.
- Development of database *structuring* is often *ruled by* the intentions for the *utilization* of the database. These intentions are based on main functionality of the database. Normalization and later denormalization shows that functional requirements may be conflicting.
- *Discretization* of data or *conversion of continuous data to discrete data* will lead to different behavior and different query facilities of databases. Discretization may be based on time, space and other abstractions which may vary depending on the point of view the specific application is considered at the time of the development.
- Database developers make their assumption on the *name space* to be used. Name spaces depend on the concepts used in the application area.
- The chosen *modeling language* imposes a number of *restrictions* to structuring of the database. Some of these restrictions are unnatural, do not apply to the implementation platform, and lead to introduction of artificial types that do not have a meaning in the application area.
- The *scope of data representation* is often concentrated on the scope of the user at the business user level. This restriction leads to representation of macro-data which are comprehensions of micro-data that must have been used in the database.
- *Data abstractions* are often used instead of basic data. Since abstractions ease querying systems are faster. At the same time, modification might be very complex. If abstractions are used then the application must be remodeled to basic data structuring supported by view processing for computation of data abstractions.
- *Optimization* of structuring to performance and tuning uses *normalization techniques*. Since the same set of constraints may lead to different normalized structures, optimization decisions must be made explicit.

One kind of difficulties of the database integration problem is caused by the development culture which does not force these implicit assumptions to be accessible.

2 Known Approaches to Structural Integration of Databases

Structural integration of databases can be defined by the triple $\mathfrak{I} = (\mathcal{G}, \mathfrak{S}, \mathcal{M})$ consisting of a global database schema $\mathcal{G}$ (over a language $\mathcal{A}_\mathcal{G}$), a collection $\mathfrak{S}$ of local database schemata $\mathcal{S}$ (over a language $\mathcal{A}_\mathcal{S}$), and a mapping between $\mathcal{G}$ and $\mathfrak{S}$.

Database integration has been discussed over a long period of time. A negative result that is often neglected in research and applications is the following:

Proposition 1 *(Convent 1986) The problem whether databases can be integrated is* undecidable.

It is, however, an observation often made in applications that these application databases can be integrated. We find a heuristic assumption:

Heuristic assumption 1 Applications are integratable since there is no reason for non-integration.

This assumption is based on a rather simple precondition of enterprises:
The application is already integrated in general. Some parts of the application are separated and supported by different information systems. Their collaboration is explicitly supported or can be explicitly supported.

Also, it is often observed that existing database systems can often be integrated. The typical approach is to find a common superschema that contains the database structures. If such a schema exists then database can be treated by the schema, i.e., a part of the instance is made visible to the user as a *virtual schema*. However, this advantage has several drawbacks.

1. It may not be possible to update the databases through the integrated database.
2. Second, the average user cannot use the whole schema due to its size. Therefore, he/she cannot understand the impact of his/her procedures.

Databases to be integrated can be considered to be views of the integrated database. Therefore, the integrated database may support the entire application. In the past, three approaches (Calvanese, Giacomo, Lenzerini, Nardi & Rosati 1998, Thalheim 2000) have been worked out to treat integrated databases:

Global-as-view integration (GAV): The integrated database is virtual. In reality, the local databases are still running on their own. There are no common functions or queries. GAV supports a *client-driven integration* and bottom-up development and extension of local source systems. The GAV approach reduces query processing to view processing.

Local-as-view integration (LAV) : (Lenzerini 2002) The database integration allows us to build a data warehouse containing all data of the local application. The data of the local application corresponds to virtual or materialized views of the global database. Any change of the local data is harmonized with the global data if the change is going to be supported. LAV supports *source-driven integration* of applications and top-down design of applications by incremental addition of new sources. The global integration of all local databases supports consistency of all data and rejects any wrong modification of the database in a very early stage. The integration effort is, however, rather high. LAV often forces a reconsideration of the local schema. Some of the local applications must be redeveloped and reimplemented.

View cooperation: (Thalheim 2000) Database cooperation is supported by exploiting the import/export facilities of the local databases. Each of the local database systems provides a

number of views to the other databases. These views are either export views or import views of the collaborating databases. The schema of an importing view of the importing system contains the schema of an exporting view of the exporting system. View cooperation combines the local-as-view and the global-as-view approaches while maintaining their advantages. The mapping of the databases is similar to LAV mappings.

The view cooperation approach is at the same time the most general approach. We may immediately derive the following corollary:

Proposition 2 *Local-as-view integration and global-as-view integration can be expressed through view cooperation expressions.*

Often full integration is not the aim. The aim is to achieve consistency. In this case the views should be (pair-wise) consistent via some translation mechanism: databases cooperate. This *database cooperation* mechanism is based on the construction of functions mapping parts of the view instances on parts of the other view instances. The next generalization step is to build the interface mechanism as a whole through *scripts*.

3 Towards Database Collaboration

3.1 Database Cooperation

The integration methods discussed above use inheritance of IsA-relationship types: all attributes and operations of a metaclass are propagated to their subclasses unless overridden explicitly by a subclass. Explicit definition of the cooperation functions is more general. We say that one view A *dominates* the view B if a set of formulas exists such that the types of the view A can be embedded into B . Thus the view integration problem determines whether a minimal schema exists for a collection of views such that the schema dominates each of the views.

We say that the views A, B *cooperate* via the (partial) functions f_A, f_B defined on $SAT(A), SAT(B)$

$f_A : SAT(A) \multimap SAT(B)$
$f_B : SAT(B) \multimap SAT(A)$

if for each $v_A \in SAT(A)$, $v_B \in SAT(B)$ the functions $f_A(v_A)$, $f_B(f_A(v_A))$, $f_B(v_B)$, $f_A(f_B(v_B))$ are defined and $f_B(f_A(v_A)) = v_A$, $f_A(f_B(v_B)) = v_B$.

The functions for view cooperation can be composed of functions in different parts of the view. Generally speaking, views cannot be completely mapped onto each other. Therefore, to decide whether two views cooperate we need to complete the following tasks:

1. Find parts of the two views which are candidates for cooperation.
2. For these candidates find the corresponding cooperation functions.
3. Compose the view cooperation functions.

In order to establish whether parts of a view cooperate with other parts we use semantic information about the views.

The question as to whether views can be integrated or can cooperate can be answered if semantics of the views are well-defined. Integrity constraints can be used for this purpose. If one of the subset relationships is valid then the corresponding types can be embedded into their supertypes. This approach can be extended to view cooperation as displayed in Figure 1.

Assume schemata S_1, S_2 and selectors sel_1, sel_2 defined on S_1, S_2. The views V_1, V_2 can be defined by the given selectors. Furthermore, take two (S_1, S_2, sel_1, sel_2) functions

$f_1 : SAT(V_1) \multimap SAT(V_2)$
$f_2 : SAT(V_2) \multimap SAT(V_1)$.

We notice that $SAT(V_i) = sel_i(SAT(S_i))$ for $i = 1, 2$. For given databases db_1, db_2 on S_1, S_2, selectors sel_1, sel_2 and the corresponding views, two functions f_1, f_2 match if

$f_1(sel(db_1)) \cup_{S_2} db_2 \in SAT(S_2)$ and
$f_2(sel(db_2)) \cup_{S_1} db_1 \in SAT(S_1)$.

Two (S_1, S_2, sel_1, sel_2) functions f_1, f_2 are view cooperation functions if the functions match with regard to all $(db_1, db_2) \in (SAT(S_1), SAT(S_2))$.

The problem concerning whether (S_1, S_2, sel_1, sel_2) functions exist is a generalization of the view updateability problem for $S_1 = S_2$ and $sel_1 = sel_2$. In this case, the function f_1 is an embedding function.

The *global view cooperation problem* determines whether view cooperation (S_1, S_2, sel_1, sel_2) functions exist. The *restricted view cooperation problem* determines whether there exist restricted view cooperation (S_1, S_2, sel_1, sel_2)-functions id, id, i.e. for all $(db_1, db_2) \in (SAT(S_1), SAT(S_2))$ with $sel_i(db_i) = (sel_j(db_j))$ and $i, j \in \{1, 2\}, i \neq j$.

Two views defined on S_1, S_2, sel_1, sel_2 are said to be *consistent* if view cooperation functions exist.

View cooperation and integration can be based on the construction of subtype/supertype hierarchies, e.g., for the integration of conceptual graphs. This approach is based on strong semantics for cardinality constraints. The theory of extended entity-relationship models can be used to derive conditions for view cooperations. It is well known (Thalheim 2000) that the subtype/supertype hierarchy must be consistent with the view cooperation schema.

3.2 Application of Cooperation to Multi Database Systems and Database Farms

Distributed database systems are based on local database systems and follow a certain integration strategy. Integration is based on total integration of the local conceptual schemata into a global distribution schema.

Open multi-database systems are a variant of distributed systems with a distribution schema that does not integrate the local systems but supports an identification of the database systems and their data. Database system integration has been tackled on the basis of federated database systems. Their architecture is similar to the one in Figure 2. The container systems do not contain any additional programs. The global communication and farming system is a simple transfer system in this case. Federated database systems are distributed database systems which use local database systems for support of global applications. Federated database systems have not yet succeeded in practical applications. The main reason is the technical difficulty. Federated systems must be supported by sophisticated integrity maintenance, powerful communication and transaction protocols and by systems for automatic decomposition and generation of functionality.

Database farms (Yigitbasi, Thalheim, Seelig, Radochla & Jurk 1999) are generalizing and extending these approaches. Their architecture is displayed in Figure 2. Farms are based on the codesign approach (Thalheim 2003) and the information unit and container paradigm:

Information units are generalized views. Views are generated on the basis of the database. Units

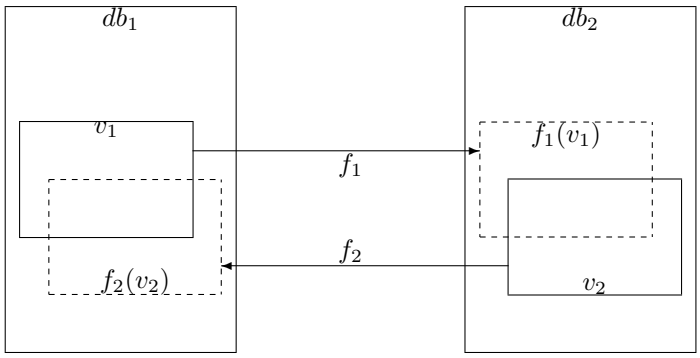

Figure 1: View Cooperation in Databases

are views extended by functionality necessary for the utilization of view data. We distinguish between *retrieval information units* and *modification information units*. The first are used for data injection. The later allow to modify the local database.

Containers support the export and the import of data by bundling information units. Units are composed to containers which can be loaded and unloaded in a specific way.

The global communication and farming system provides the exchange protocols, has facilities for loading and unloading containers and for updates of modification information units.

We do not want to integrate entirely the local databases but provide only *cooperating views*.

Database farms are more complex to design. The computational support is entirely based on classical database technology. Therefore, if we are able to design such integrated system farms the management is feasible.

3.3 Application of Cooperation to Incremental Database Systems

Integration of systems can be based on *hub points* at which systems may plug and have the same behavior. Information-lossy integration may be based on abstraction if the information loss is restricted to those data which are not of interest in the other application or which may be computed by the other application.

The theory of hub types supports incremental evolution of database systems (Raak 2001) which is a specific form of database system evolution. Facility management systems are typical application systems for which incremental evolution could be the ultimate solution. Typical for such applications is the long lifespan of some of the objects. Those objects have a long history of change. The architecture of facility management is sketched in Figure 3. We use *auxiliary databases* for support of the facility management system. Such data provide help information, information on regulations, information on customers, information on suppliers, etc.

Incremental evolution is thus supported by:

Injection forms enable to inject data into another database. The forms are supported by views and view cooperation approaches. Data injected into another database cannot be changed by the importing database system. The structuring $(\mathcal{S}^{inject}, \Sigma_{\mathcal{S}})$ of the views of the exporting database system is entirely embedded into the structuring $(\mathcal{S}^{'}, \Sigma_{\mathcal{S}'})$ of the importing database system. The functionality $(\mathcal{O}^{inject}, \Sigma_{\mathcal{O}})$ of the views of the exporting database system is partially embedded into the functionality $(\mathcal{O}^{'}, \Sigma_{\mathcal{O}'})$ of the importing database system by removing all modification operations on the injected data. These data can only be used for retrieval purposes.

Insertion forms enable in insertion data from the exporting database into the importing database. These data can be modified. The structuring $(\mathcal{S}^{insert}, \Sigma_{\mathcal{S}})$ and the functionality $(\mathcal{O}^{insert}, \Sigma_{\mathcal{O}})$ of the views of the exporting database system are entirely embedded into the structuring $(\mathcal{S}^{'}, \Sigma_{\mathcal{S}'})$ and the functionality $(\mathcal{O}^{'}, \Sigma_{\mathcal{O}'})$ of the importing database system.

3.4 Database Collaboration in the Washer Approach

The Cottbus database and information systems research group developed in one of its industry projects (Radochla & Thalheim 1999) a specific extension of the view cooperation approach: The Washer[1] approach is based on view cooperation and explicit modeling of coordination among several databases. The general architecture is depicted in figure 4. The washer is a tool that manages the collaboration based on the the coordination profile. The coordination profile is specified by a *coordination contract*, a *coordination workspace*, *synchronization profile*, *coordination workflow*, and *task distribution*.

Coordination is based on a coordination contract. The *contract* consists of

- the coordination party characterization, their roles, rights and relations,
- the organization frames of coordination specifying the time and schema, the synchronization frame, the coordination workflow frame, and the task distribution frame,
- the context of coordination, and
- the quality requirements (ubiquity, security, interpretability, consistency, view consistency, scalability, durability, robustness, performance) for coordination.

We distinguish between the frame for coordination and the actual coordination. Any actual coordination is an instance of the frame. Additionally, it uses an infrastructure. The contract specifies the general properties of coordination. Several variants of coordination may be proposed. The formation of a coordination may be based on a specific infrastructure. For instance, the washer may provide a workspace and additional functionality to the collaborating partners.

[1] A washer is a ring of metal between a nut and a bolt, or between two pipes to make a better and tighter joint.

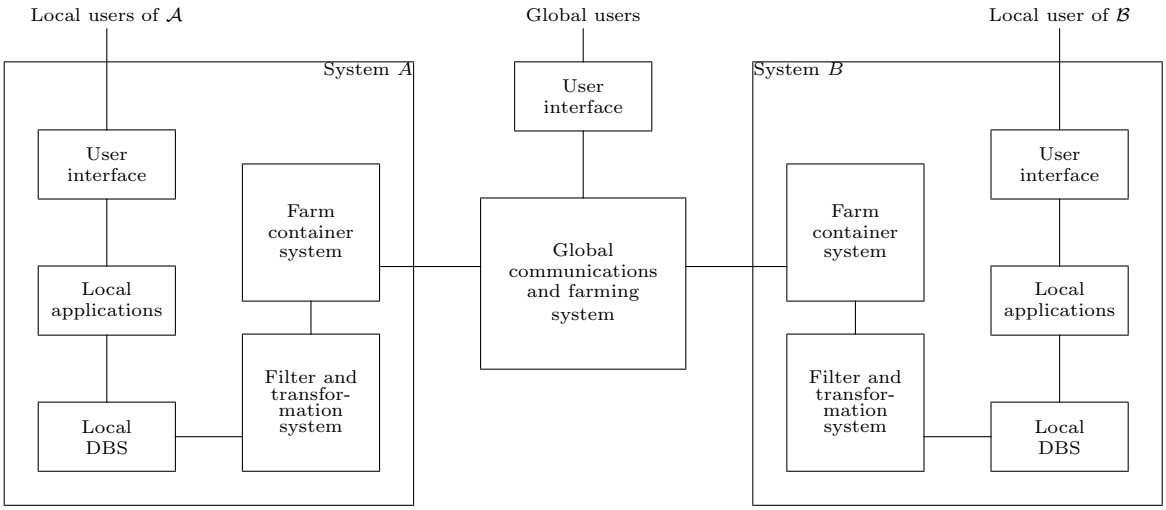

Figure 2: Database Systems Farm

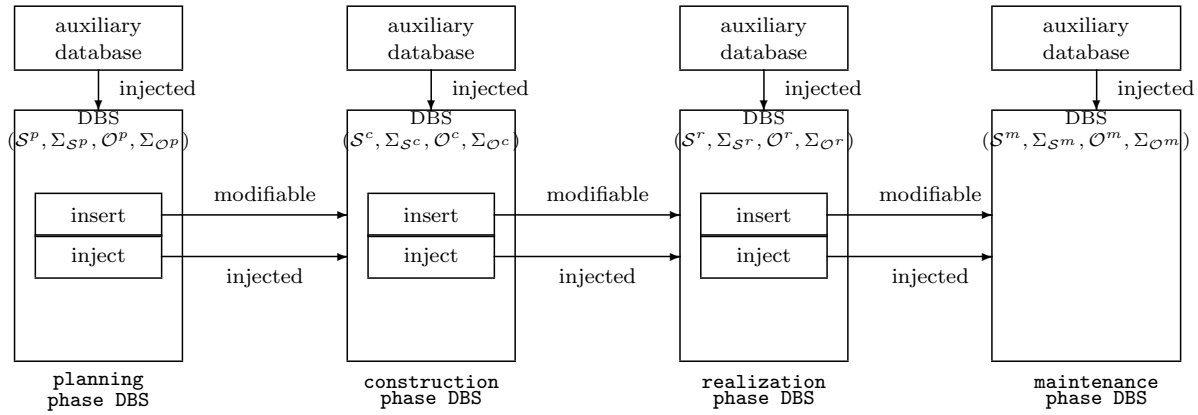

Figure 3: The General Architecture of Incremental Evolution of Database Systems

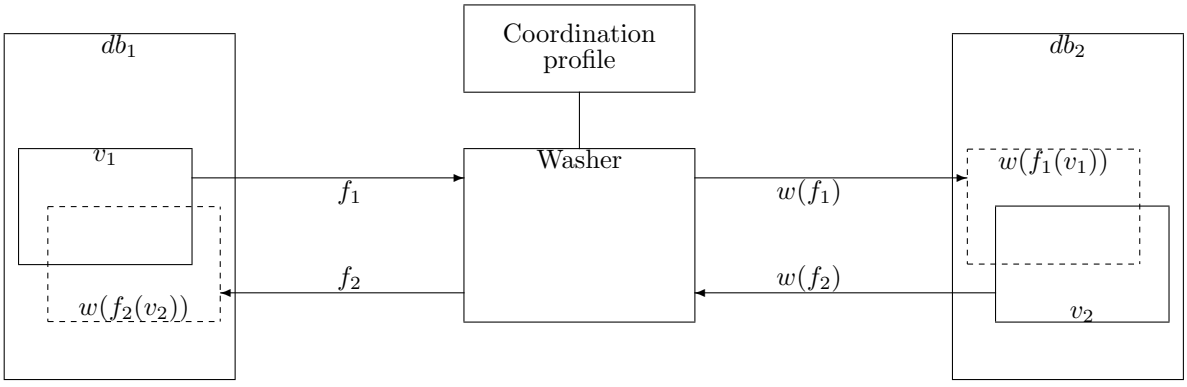

Figure 4: The Washer Approach to Collaboration of Databases

Formation			Parties				Organization			Infrastructure	
Contract	Life-span	Contract variant	Properties	Roles	Rights	Relations	Synchronization	Workflow	Task distribution	Workspace	Support
...	...	...	...	...	...	...	...	...	...	...	...

Table 1: Coordination Profile

The coordination profile is specified by the frame shown in table 1.

Collaboration is based on *communication*, *cooperation*, and *coordination*. Cooperation specification follows a similar approach. It is restricted by the *cooperation contract* that specifies

- the services provided, i.e., informational processes consisting of *views* of the source databases, the *services manager* supporting functionality and quality of services, and the *competence of a service* manifested in the set of tasks that may be performed, and
- requirements for *quality of service*.

Communication contracts specify the collaboration architecture and the style of exchange. Typical collaboration architectures are for example proxy collaboration, broker-customer, or publisher-subscriber collaboration. The exchange frame generalizes protocols and is defined by

- *collaboration style* specifying the supporting programs, the style of collaboration and the collaboration facilities, and
- *collaboration pattern* specifying the roles of the partners, their responsibilities, their rights and the protocols they may rely on.

In a similar fashion we may specify the communication profile and the cooperation profile. The project has led to an integration of SAP R/3, of several Oracle databases and of OLAP functionality of SAS.

4 Derivation of Collaboration Strategies

Based on a number of projects and the results of theory development we propose an approach that is both based on

- *separation of functionality* for those parts of the application that are developed or used in a separate fashion and
- *integration of those parts of the application which must be obligatory combined*.

Based on this twofold approach, a data warehouse can be developed that

- supports all tools in a separating fashion,
- supports versioning of development results, and
- supports stepwise enrichment of object sets whenever they have been used by another tool supported by the warehouse.

4.1 Derivation of Collaborating Sub-Structures

Integration of structures can be based on a repository, intensional knowledge from the application area and by applying the following heuristic procedure (Kalinichenko 1999):

1. *Separation of the global schema* by functional aspects of the application in the case that full schema integration is infeasible or impossible;
2. *Transformation of schema specification* by compiling the schema information of databases to be integrated into an intermediate schema;
3. *Enrichment of schema specification* by repository, metadata or concept information;
4. *Development of data warehouse kernels* consisting of cooperating views;
5. *Derivation of requirements for wrappers* which support import and export.

Each of these steps may fail. If the step does not lead to a satisfying result then we may use a number of heuristics in order to overcome the difficulties.

4.1.1 Separation of Source Schemata in the Case of Partial Integrability

Full integration of a set of source schemata might be not feasible, not achievable or not intended in the application. Instead we might be interested in cooperation or collaboration of several applications and try to support a consistent exchange and maintenance of commonly shared data. The first step concentrates on elicitation of integration requirements.

Scope of separation.
Since we do not aim for the construction of an entirely integrated global schema but also support cooperation of source schemata we develop an approach that supports partial integration. *Partial integration* must cover at least all *global features* and *exchange of commonly shared data*. If all features of the application are local then only commonly shared data are treated by an integration architecture. We consider data exchange functions to be a part of the set of global features.

The set of features of an application and the set of exchange features for commonly shared data is now collected and considered.

Steps of separation.
The *separation step* uses information on the functionality of the application. Each tool uses a collection of functions. Each of these functions uses a number of types of the schema, called the *environment* $\mathcal{E}(f)$ of the function f. We distinguish between input, output and modification types for each function. ($\mathcal{E}^I(f)$, $\mathcal{E}^O(f)$, $\mathcal{E}^M(f)$). We may now consider the environment for collections of functions used in an application feature. Given a feature $\mathcal{F}$ that uses the set of functions $\mathcal{F}$. The environment for the feature is defined by the union of the environments of functions f from $\mathcal{F}$.

The separation into input, output and modification types allows to develop a more sophisticated theory of collaboration of source schemata. The view cooperation approach nicely supports this separation. In the sequel we do not use this separation. The separation adds some technical details that are not difficult to treat. The separation allows to develop a theory of finer granularity. For our purposes, we develop first the general approach without separation. The general approach can be understood as a pessimistic approach that detects all potential types for integration. The separation computes only those types that are necessary for integration.

We find however commonly shared data of all or at least two applications. The corresponding data types are locally defined. They represent however the same set of things in the application. Therefore, we may now use such potential exchange types.

The separation step extracts obligations for integration. We may proceed using the following substeps:

Scoping of sub-schemata for global features: We form sub-schemata for each of the feature collections. The sub-schemata may be defined through views on the source schemata. Views are defined as queries on types of the source schema.

The environment $\mathcal{E}(\mathcal{F}, \mathcal{S})$ of a global feature $\mathcal{F}$ and a source schema $\mathcal{S}$ is the set of types used in views supporting the functions in $\mathcal{F}$ within the source schema $\mathcal{S}$.

Extending the environment by enforcement context: The environment of global functions $\mathcal{E}(\mathcal{F}, \mathcal{S})$ forms a sub-schema in the source schema $\mathcal{S}$. Changes of data within the environment must

be locally consistent. Therefore, integrity constraints must be considered as well. Most of the integrity constraints cannot be enforced locally only for the class of data of a type T. For instance, an inclusion constraint $T_1[X_1] \subseteq T_2[X_2]$ extends the integrity enforcement environment of T_i by T_j for $i, j \in \{1, 2\}$.

Let $\mathcal{E}_\Sigma(T)$ be the integrity constraint environment of a type T for a set of integrity constraints Σ. We extend now the environment $\mathcal{E}(\mathcal{F}, \mathcal{S})$ by the integrity enforcement environment for each type in $\mathcal{E}(\mathcal{F}, \mathcal{S})$. Let $\mathcal{E}_\Sigma(\mathcal{F}, \mathcal{S})$ be this environment.

Developing a view suite for sub-schemata: The environments $\mathcal{E}_\Sigma(\mathcal{F}, \mathcal{S})$ are now used for deriving the views necessary for the support of global features.

If we prefer the pessimistic approach then we obtain a sub-schema $\mathcal{V}(\mathcal{S})$ of a given source schema $\mathcal{S}$ by considering all types of $\mathcal{E}_\Sigma(\mathcal{F}, \mathcal{S})$ over the set of global features. If we are interested in a more sophisticated approach then we obtain three sub-schemata $\mathcal{V}^I(\mathcal{S})$, $\mathcal{V}^O(\mathcal{S})$ and $\mathcal{V}^M(\mathcal{S})$ of the source schema $\mathcal{S}$.

Collection of integration obligations: The sub-schemata $\mathcal{V}(\mathcal{S})$ form now the integration obligations for the set of global features.

We may interactively add to these sub-schemata other types of the source schema. This pragmatical extension allows us to consider potential integration of other databases if the application is going to be extended.

This step results in views on the source schemata that are now to be integrated. Their cooperation is tight, i.e., the collaboration of applications is based on a full integration of the types or on development of washers for cooperating schemata.

In the sequel we concentrate on the case that full integration can be achieved. The case that collaboration facilities must be used is similar.

4.1.2 Transformation

The *transformation step* is based on the extraction of a logic theory supporting reasoning on name spaces between the types of the schemata under consideration. Name spaces of the schemata under consideration may be compared by their "similarity" on the basis of synonym and homonym equalities and inequalities. Equalities and inequalities are enriched by *plausibility coefficients* that measure the confidence of the actual equality or inequality. The confidence measure obeys properties of t-norms used for Fuzzy logics. Additionally we may use *context* for the confidence measure. The logical theory uses a number of comparison predicates:

Synonyms $S \approx T$ specify that two names in schemata under consideration have the same meaning or semantics.

Synonyms may also be based on identification expressions. Identification of things may be different in different applications although objects relate to the same thing of the reality, e.g., student number and student identification data.

Synonym associations can also be developed for query expressions defined on two schemata. Typically, such synonym expressions are semantic conversions for domains, e.g., converting fuel consumption data used in Germany to fuel consumption data used in the US.

Synonym expressions may be generalized to *data integration mediators*. These can be stored in a database that extends the current database by integrating mediators. We use a similar mechanism on the basis of *extended identification*.

Synonym associations may be combined with a *preference rule* stating which type name is going to be used if the two synonyms are mapped to one type in the integrated schema.

Homonyms $S \between T$ describe structural equivalence combined at the same time with different meaning and semantics. Homonyms may be simply seen as the negation or inversion of synonymy. Since the confidence level may be different we prefer to use homonyms as well.

Hyponyms **and** hypernyms $S \preccurlyeq T$ hint on subtype associations among types under consideration. The type T can be considered to be a more general type than S and the integration of the two types leads to an explicit subtype association in the integrated schema.

Overlappings **and** compatibilites $S \overset{\exists}{\Cup} T$ describe partial similarities. These similarities can be treated by introduction of generalizing supertypes.

Explicit representation conflicts supports the application of conflict resolution strategies such as renaming, homogenizing representations, homogenizing types, application of extended database operations such as extended join and other homogenizing operations. Representation conflicts may lead to integration *obligations* for interactive resolution of those conflicts.

Abstraction similiarities $\mathfrak{S} \approx \mathfrak{T}$ support the development of name space transformations. Subschemata may be abstracted into abstract types. Abstract types of different schemata may be associated. We may explicitly use this meta-similarity or meta-heterogeneity for association or separation of sub-schemata.

Additionally, we may require that the equality logic is invariant with the structuring given in the schemata. The metrics can be structurally based. The deduction system in (Thalheim 2000) for inclusion and exclusion constraints may easily be generalized to a deduction system for reasoning on synonyms, homonyms, hyponyms and potential supertypes. Since the logical system may be become too granular we may use a threshold logic for more abstract reasoning on potential integrability.

Structural transformation also removes *pragmatic assumptions* made during database development. Typical such assumptions are the *objectification* and the introduction of entity types, relationship types, cluster types, and attribute types. In one application, for instance, it might be useful to use entity types instead of attribute types because of limitations of the platform. If the modeling methodology uses the 'dividing range' then the objectification decision is explicit. Older methodologies do not use this conception and/or use only atomic attributes. In the latter case, objectification must be based on the elicitation of additional knowledge on the application area.

The structural and semantical information can be used for *preintegration* and *comparison* of schemata. Preintegration is based on a strategy of the order of integration. The best order known so far is the inductive order following the order of structural types, i.e., starting with domains, followed by attribute types, followed by entity types and then finally by relationship types depending on their order. During comparison, naming and structural affinities and conflicts are derived, synonyms are unified to common names,

homonyms are separated based on name extensions (e.g., prefixes), hyponyms are used to form hierarchies within the schema, and overlappings are used for developing generalizations.

The result of the step is a *mediating and separating ontology* of types used in the schemata under consideration. This ontology is associated with queries for extraction of the corresponding concepts of the databases. It supports the derivation of mappings for cooperating views.

This mediating ontology can be extended to generation of the global schema $\mathcal{G}$ for the triple $\mathfrak{I} = (\mathcal{G}, \mathfrak{S}, \mathcal{M})$, the collection $\mathfrak{S}$ of local database schemata $\mathcal{S}$ (over a language $\mathcal{A}_\mathcal{S}$), and the mapping between $\mathcal{G}$ and $\mathfrak{S}$. Given an equality and inequality theory $\Gamma_{\approx,\emptyset,\preccurlyeq,\Cup}^{\exists}$. We can derive the weakest similarity relation $\backsimeq$ expressing that two types are definitely equal and the strongest similarity relation $\approxeq$ expressing that two types are potentially equal, i.e., there is no objection against their equality. These two similarity relations may be used for automatic derivation of the

- *weakest global integration schema* $\mathcal{G}_{\backsimeq} = (\bigcup \mathfrak{S})_{|\backsimeq}$ and
- *strongest global integration schema* $\mathcal{G}_{\approxeq} = (\bigcup \mathfrak{S})_{|\approxeq}$.

The mappings $\mathcal{M}_{\backsimeq}$ and $\mathcal{M}_{\approxeq}$ are constructed in a similar way if a preference relation for the choice of the integration type is provided.

The *intermediate global schema* $\mathcal{G}$ is usually a schema that is weaker than the strongest integration schema and stronger than the weakest integration schema. The decisions which strictness for integration is applied will be derived in the following steps.

4.1.3 Enrichment

Necessity of enrichment due to forgetful database development.
Since database applications often have been developed by applying a specific scope, by restricting to a number of aspects, by using implicit assumptions discussed above and by pragmatical decisions, integration of specifications might not be possible. The integration conflicts are mainly based on the database system development itself. Forgetfulness of database system development can be reduced if the development teams agree on a common repository and on the same style of making design decisions. Moreover, information systems have been developed following different paradigms such as relational database technology, object-relational database technology, object-oriented database technology and file storage technology. Older tools use the latter technology. Application systems may follow an event-based computation style or a state-changing computation paradigm.

Therefore, beside intentional forgetfulness we observe also paradigm-based forgetfulness. If we face the need of system integration then we need to know the context of the system development and the style of system computation. Forgetful development is not only a part of the development culture but is also caused by different treatment of the theory used. Different theories use also different name spaces. These name spaces are based on a large variety of notions used in the application area.

4.1.4 Development of the Data Warehouse Kernel

The general architecture of the data warehouse uses the separation of source schemata $\mathcal{S}$ into

- sub-schemata $\mathcal{V}(\mathcal{S})$ that must be integrated and sub-schemata that coexist together with other sub-schemata of other applications and
- sub-schemata $\mathcal{S} \setminus \mathcal{V}(\mathcal{S})$ that are not under consideration for integration.

Data warehouses integrating several applications contain all data of the applications. The data is separated into data that belongs to one and only one application and data that belongs to several applications. The relation among the commonly shared data forms the skeleton of the application. This skeleton may be rather complex. We may simplify the skeleton by developing a general integration kernel and by associating the integration kernel with the data that are commonly used. In the simplest case the skeleton forms a star structured data warehouse as pictured above. The kernel coincides with the data structure of the integrated data structure of the data warehouse. This simple structure is often not achievable. We may, however, use a *surrogate kernel* by introducing a number of artificial types that may not have a meaning in the application but nicely support integration and consistent management of data in the data warehouse.

The development of the data warehouse kernel for integrating n tools is based on a number of steps:

Development of abstractions within the schemata:
Since large schemata are hard to understand, we simplify the schema by applying schema abstraction and clustering techniques. Clustering is a recursive procedure that constructs shells of main types. Depending on the adherence, types of a shell may be clustered to one type for external representation and collaboration issues.

Development of a hub meta-structure and simplification of the skeleton: The skeleton of the integrated schema is reconsidered for detection of generalizable structures, for transfer of bilateral associations to trilateral, then of trilateral associations to 4-lateral etc. and finally of (n-1)-lateral associations to n-lateral. If the i-lateral association of source schemata can be easily maintained then we do not transfer them to higher laterality.

The transfer to higher associations is based on the introduction of surrogate identifiers that can be mapped to the identification of types. These surrogate identifiers imply two surrogate functional dependencies relating the surrogate with the original identification.

The derivation of surrogates is recorded for automatic derivation of integrity constraint management of the surrogate functional dependencies.

This transformation of the data warehouse schema forms a hub-like skeleton. This skeleton has the advantage that changes of data by one tool can easily mapped to changes of data for other tools.

Development of a surrogate generation facility:
Surrogate identifiers can be simpler handled if the generation mechanism is well-specified. The generation mechanism may be used for derivation of indexes, for derivation of direct search facilities, and for collection of data suites for transfer of data from the warehouse to the tools.

Development of version management: Version management becomes a major obstacle if the local applications are intensively used from time to time. We can integrate the version mechanism into the surrogate value generation if versioning is going to be hierarchical. In this case we use an identifier suffix extension as the version number.

4.1.5 Derivation of requirements for wrappers

Finally, requirements for *wrappers* that support integration, import and export of data are developed. These wrappers support the following tasks:

Input/output/modification interfaces: Data in the data warehouse and data used, modified, and generated by local applications must be constantly harmonized. The import/export of data from the warehouse to the tools is based on tool enactment. Whenever a tool is activated the data suite necessary for support of the local tool is either transferred from the warehouse or a new data suite for the tool is automatically copied to the warehouse. After the tool is deactivated a new version is inserted into the warehouse.

Consistency control for data suites: Data suites are kept together for all active local tools. Consistency of kernel data common to all applications is constantly maintained by triggers and monitors observing the state of the database.

Consistent playout of data generated by some of the tools: A challenge to the data warehouse that can only be partially resolved is the consistent playout of data suites by several tools. To support this challenge, special playout wrappers may be developed that show the results of simulations of several tools. An XML based interface might support the consistent playout. This solution is not a general one but works only under some restrictions applied to the local applications.

The wrappers thus support the hub-based integration within a data warehouse. At the same time wrappers may be used to support consistency management. Consistency management and constraint enforcement are one of the most difficult database design and database development issues. Integrity enforcement is usually supported by

decomposition (normalization) of structures to such structures which allow a simple integrity enforcement (mainly on the basis of keys, referential integrity constraints, and domain constraints), or by

extensions of database operations that maintain consistency of the database while have the same effect as the operation that has been extended, or by

special programs for integrity enforcement such as triggers, assertions and stored procedures, or by

transaction management that rejects all those modifications of the database that leads to inconsistent states, or by

application interfaces keeping consistency of those data suites which may be falsified within a program that uses insert, delete or update operations.

All five approaches have their disadvantages. Decomposition approaches may fail due to the complexity of the integrity constraint set. Extensions of operations cannot be computed in all cases. Special programs supporting integrity management must be combined with control and scheduling facilities in order to avoid avalanches. A solution for the last problem is not known yet. Transaction management might be too restrictive and too pessimistic. The last approach to integrity management is feasible as long as the application is modifying the database only through the application interfaces and the interfaces are well developed.

In our case we can use the last approach. We can extend the hub architecture in a way that it maintains integrity constraints and versions of data suites.

4.2 Collaboration Warehouses

Objects may be developed by each application on their own. Objects belonging together to one version of the development process are called an object suite. A *suite* consists of a set of elements, an integration or association schema and obligations requiring maintenance of the association. A suite will be accepted by the collaborating database set

- by transforming the suite into a set of objects within the data warehouse,
- by adding to the identification of objects their object suite identification,
- by extracting the identification tree of the suite identification, and
- by associating the object suite with the application working so far with the suite.

The identification tree of an object suite is called *hub kernel*. Through this hub kernel the object suite can be reestablished.

An application may either call an existing object suite or insert a new object suite into the data warehouse. If an application calls an object suite and the object suite has been developed by another application then the object suite will be also associated with the new application. Any application may directly change those parts of the object suite which exclusively belong to the structures and attributes of this application. If data in an object suite are changed that belong to several applications then an explicit cooperation function must support the consistency of the parts of the object suite.

The hub kernel may not be changed by any application. It is only possible to delete the entire object suite if none of the applications is using one the objects in the suite.

The collaborating database suite is based on an explicit specification of the *cooperation pattern*, of the *coordination pattern* and of the *communication pattern* of object suites. These patterns are composed to a general collaboration contract. This contract specifies which application may perform which modification of an object suite under which conditions at which time.

5 Conclusion

Database integration is not achievable in general. In most applications a full integration of database systems is not an aim of the development. We develop a novel approach to database systems collaboration that allows to develop a suite of database systems which are collaborating. The collaboration is based on explicit specification of cooperation, coordination and communication. The specification has already been applied in some of our projects and may thus considered to be practical and applicable. At the same time we know that our proposal will not provide the ultimate solution to all database integration problems. It provides, however, a way to integrate systems as much as possible.

References

Abiteboul, S., Cluet, S. & Milo, T. (1997), Correspondence and translation for heterogeneous data, *in* F. N. Afrati & P. Kolaitis, eds, 'Database Theory - ICDT '97, 6th International Conference, Delphi, Greece, January 8-10, 1997, Proceedings', Vol. 1186 of *Lecture Notes in Computer Science*, Springer, pp. 351–363.

Anthony Hunter, B. N. (1998), 'Managing inconsistent specifications: Reasoning, analysis, and action', *ACM Transactions on Software Engineering and Methodology (TOSEM)* **7**(4), 335–367.

Beeri, C. & Milo, T. (1999), Schemas for integration and translation of structured and semi-structured data, *in* C. Beeri & P. Buneman, eds, 'Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings', Vol. 1540 of *Lecture Notes in Computer Science*, Springer, pp. 296–313.

Bernstein, P. A. & Rahm, E. (2001), 'A survey of approaches to automatic schema matching', *VLDB Journal* **10**, 334–350.

Calvanese, D., Giacomo, G. D., Lenzerini, M., Nardi, D. & Rosati, R. (1998), Information integration: Conceptual modeling and reasoning support, *in* 'Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, August 20-22, 1998, Sponsored by IFCIS, The Intn'l Foundation on Cooperative Information Systems', IEEE-CS Press, pp. 280–291.

Christine Parent, S. S. (1998), 'Issues and approaches of database integration', *CACM* **41**(5), 166–178.

Cluet, S., Delobel, C., Siméon, J. & Smaga, K. (1998), Your mediators need data conversion!, *in* L. M. Haas & A. Tiwary, eds, 'SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA', ACM Press, pp. 177–188.

Conrad, S., Saake, G. & Sattler, K.-U. (1999), Informationfusion - Herausforderung an die Datenbanktechnologie, *in* 'Proc. BTW', Springer, pp. 307–316.

Convent, B. (1986), Unsolvable problems related to the view integration approach., *in* 'ICDT'86', Vol. 243 of *Lecture Notes in Computer Science*, Springer, pp. 141–156.

Kalinichenko, L. A. (1999), Compositional specification calculus for information systems development, *in* J. Eder, I. Rozman & T. Welzer, eds, 'Advances in Databases and Information Systems, Third East European Conference, ADBIS'99, Maribor, Slovenia, September 13-16, 1999, Proceedings', Vol. 1691 of *Lecture Notes in Computer Science*, Springer, pp. 317–331.

Lee, M.-L. & Ling, T. (1995), Resolving structural conflicts in the integration of entity-relationship schemas., *in* 'Proc. ER'95', Vol. 1021 of *Lecture Notes in Computer Science*, Springer, pp. 424–433.

Lee, M.-L. & Ling, T. (1997), Resolving constraint conflicts in the integration of entity-relationship schemas., *in* 'Proc. ER'97', Vol. 1331 of *Lecture Notes in Computer Science*, Springer, pp. 394–407.

Lenzerini, M. (2002), Data integration: a theoretical perspective, *in* ACM, ed., 'Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems: PODS 2002: Madison, Wisconsin, June 3–5, 2002', ACM Press, New York, NY 10036, USA, pp. 233–246.

Ludäscher, B. & Gupta, A. (1999), Modeling interactive web sources for information mediation, *in* P. P. Chen, D. W. Embley, J. Kouloumdjian, S. W. Liddle & J. F. Roddick, eds, 'Advances in Conceptual Modeling: ER '99 Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, Paris, France, November 15-18, 1999, Proceedings', Vol. 1727 of *Lecture Notes in Computer Science*, Springer, pp. 225–238.

McCabe, M. C., Chowdhury, A., Grossman, D. A. & Frieder, O. (1999), A unified environment for fusion of information retrieval approaches, *in* 'Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, Kansas City, Missouri, USA, November 2-6, 1999', ACM, pp. 330–334.

Papakonstantinou, Y. & Velikhov, P. (1999), Enhancing semistructured data mediators with document type definitions, *in* 'Proceedings of the 15th International Conference on Data Engineering, 23-26 March 1999, Sydney, Austrialia', IEEE Computer Society, pp. 136–145.

Raak, T. (2001), Database systems architecture for facility management systems, Master's thesis, FHL, Civil Engineering Dept., Cottbus.

Radochla, S. & Thalheim, B. (1999), Umstrukturierung eines Data-Warehouse in ein effizientes Decision Support System, *in* F. M. F. Hüsemann, K. Küspert, ed., 'Jenauer Schriften zur Mathematik und Informatik', Vol. Math/Inf/99/16, Jena, pp. 92 – 96.

Sadri, F., Subramanian, I. N. & Lakshmanan, L. V. S. (1996), 'SchemaSQL - A language for interoperability in relational multi-database systems'.

Spaccapietra, S. & Parent, C. (1989), View integration: A step forward, reports, Lausanne University.

Thalheim, B. (2000), *Entity-relationship modeling – Foundations of database technology*, Springer, Berlin. See also http://www.informatik.tu-cottbus.de/∼thalheim/HERM.htm.

Thalheim, B. (2003), Informationssystem-Entwicklung - Die integrierte Entwicklung der Strukturierung, Funktionalität, Verteilung und Interaktivität von großen Informationssystemen, Preprint I-2003-15, Cottbus Tech, Computer Science Institut, BTU Cottbus.

Theo Härder, K. H. (2002), 'Ankopplung heterogener Anwendungssysteme an Föderierte Datenbanksysteme durch Funktionsintegration', *Informatik - Forschung und Entwicklung* **17**, 135–148.

Wiederhold, G. (1995), Modelling and system maintanance, *in* M. P. Papazoglou, ed., 'OOER'95: Object-Oriented and Entity-Relationship Modelling, 14th International Conference, Gold Coast, Australia, December 12-15, 1995, Proceedings', Vol. 1021 of *Lecture Notes in Computer Science*, Springer, pp. 1–20.

Yigitbasi, S., Thalheim, B., Seelig, K., Radochla, S. & Jurk, R. (1999), Entwicklung und Bereitstellung einer Forschungs- und Umweltdatenbank für das BTUC Innovationskolleg, *in* F. Hüttl, D. Klem & E. Weber, eds, 'Rekultivierung von Bergbaufolgelandschaften', Walter de Gruyter, Berlin, pp. 269–282.