Proc. 18th International Conference on Conceptual Modeling (ER'99, Paris, Nov.'99), LNCS, Springer-Verlag, 1999.

Design Support for Database Federations*

Kerstin Schwarz Ingo Schmitt Can Türker Michael Höding
Eyk Hildebrandt Sören Balko Stefan Conrad Gunter Saake

Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
sigmafdb@iti.cs.uni-magdeburg.de

Abstract. Federated database systems provide a homogeneous interface to possibly heterogeneous local database systems. This homogeneous interface consists of a global schema which is the result of a logical integration of the schemata of the corresponding local database systems and file systems. In this paper, we sketch the integration process and a set of tools for supporting the design process. Besides the classical database schema integration, the design process for federated information systems requires the integration of other aspects like integrity rules, authorization policies and transactional processes. This paper reports on an integrated approach to tool support of several of these integration aspects. The different integration facets are linked via the database integration method GIM allowing a high degree of automatic integration steps.

1 Introduction

The **SIGMA**$_{FDB}$ project [26] is the focus of our research in the field of federated database systems. Our common view to federated database systems is a tightly coupled approach based on the five-level schema architecture of Sheth and Larson [24]. This is motivated by the requirement of data consistency which can only be appropriately guaranteed in a tightly coupled environment. A federated database should correctly reflect the modeled real-world. Inconsistencies between the databases concerning the same real-world entities have to be removed. Correct conflict resolution builds the basis for a correct database integration.

Our research interests as well as the related implementation activities are stimulated by the following application scenarios:

- The first scenario comes from the area of factory planning. Here, different specific application programs (tools) are used for optimizing machine configuration and transport facilities in factories. These applications have been developed independently. In order to enable interoperation between these tools (in particular to support concurrent engineering) an integrated data layer is needed. A main challenge is that most tools store their data in files, each using its own way of structuring the data. Therefore, the files have to be analyzed for building a description of the file structure.

* This research was partially supported by the German State Sachsen-Anhalt under FKZ: 1987A/0025 and 1987/2527R.

- Our second integration scenario is motivated by the bio-informatics research area where we are faced with highly heterogeneous databases which contain bio-molecular data representing results of thirty years of experimental research. The integration of these partly overlapping databases promises a new quality of databases and enables new applications to access heterogeneous data in a uniform and transparent way [14].
- Our third (and most recent) integration scenario is given by the *Global-Info* project which aims at supporting digital libraries by federation services. Here, the existing systems mainly have WWW interfaces which usually restrict the access to pieces of information in a certain way.

This paper surveys our approach to designing federated database schemata. We briefly describe the tool-set $\mathbf{SIGMA}_{Bench}$ which supports the design process. The main part of this process obviously comprises the schema integration. For integrating schemata of different local systems these schemata must be available. In our application scenarios we are faced with files which do not offer a schema. Hence, the file structure must be analyzed in order to exploit a schema. The integration is performed following the GIM-Method (Generic Integration Model) [20]. For a correct integration, we have to consider both intensional and extensional conflicts in a common framework. Extensional assertions are specified by the database designer. This is a hard task for the designer and demands a detailed knowledge about the local systems. Since local integrity constraints reduce the set of possible database states, these constraints can be applied to give hints for extensional assertions between local classes [27].

A further step of the design process considers local integrity constraints as additional semantic information for schema integration. In addition, local access rights must also be considered in order to derive global access rights. Last but not least, we regard the design of global transactions and their decomposition into subtransactions as an important task in building federated information systems.

The remainder of this paper is organized as follows. In Section 2 we give an overview of the design process and current state of $\mathbf{SIGMA}_{Bench}$. Afterwards, the file analyzer and the derivation of file schemata is described. The automatic generation of extensional assertions is subject of Section 4. Section 5 discusses schema integration including integrity constraints and access rights whereas Section 6 addresses basic issues of designing global transactions.

2 Federation Design Process

Federated database systems [24] provide a homogeneous interface to possibly heterogeneous local database systems. This interface consists of a global schema which is the result of a logical integration of the involved local schemata [3]. The integration is performed in several steps being sketched in this section including file structure analysis, treatment of integrity constraints and access rights.

Fig. 1 sketches the steps of developing an integrated schema. On the first level, there are the local systems including file clusters and database systems. The assertion generator takes the local integrity constraints for the local (file)

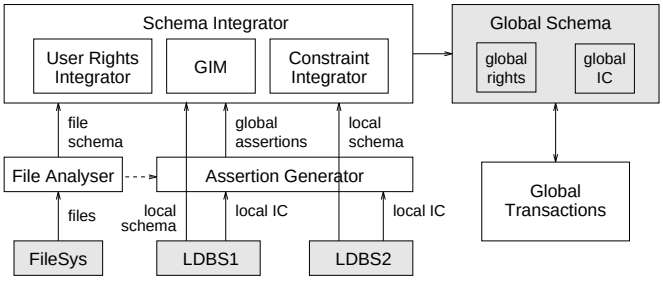

Fig. 1. Components of the Federated Schema Design Tool

schemata and generates a set of extensional assertions. Obviously, these assertions can only be complete if the local integrity constraints are fully specified. In practice, the constraints defined on a schema will probably not be sufficient to express all extensional relationships. However, they may serve as an indication for the federated database designer. The next step comprises the schema integration. Since integrity constraints and access rights influence the global schema, database integration must also comprise the integration of integrity constraints and access rights. Before integrating schemata the integrator has to complete (and probably correct) the generated global assertions. The GIM-Method deals with n-ary extensional assertions. In contrast, most other schema integration approaches are based on only binary extensional assertions which cannot express all possible situations. The result is a global schema with global access rights and integrity constraints. Each access onto the local systems over the global interface has to be done with respect to the global schema. Thus, global transactions may have different dependencies among them determined by the global assertions.

With respect to this process we implemented the federated database design tool **SIGMA**$_{Bench}$ (see Fig. 2). This tool was implemented according to a client-server architecture. Central inter-operation platform is the repository that is realized using a relational DBMS (*YARD*). Clients for dedicated tasks are developed in Java. The *file structure analyzer* comprises different tools for

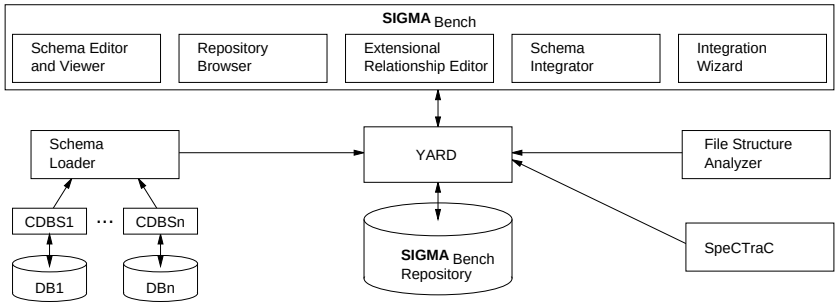

Fig. 2. Architecture of the **SIGMA**$_{Bench}$

the file structure analysis. Key words, tokens, and brackets can be found semi-automatically based on text analysis. In this way, the design of grammars describing the structure of file clusters is supported. The *schema loader* imports schemata from an Oracle database to our repository. It uses the catalogue tables of the Oracle system and currently considers classes, attributes, data types, and integrity constraints. Current implementation work extends this also to load user profiles including access rights. The *schema editor and viewer* allows the definition and graphical representation of object-oriented schemata for storing them in the repository. Of course, in "real" integration scenarios the schemata should be imported from the local databases. The main issue of the schema editor is to make schemata available for demonstration purposes. Thus, the editor offers classes, attributes, relationships, specialization, and integrity constraints. Currently three kinds of integrity constraints are supported: *uniqueness constraints* like **unique**(Book.Author, Book.Title), *attribute-constant-comparisons* (including **not null**) such as Book.Price $>$ 0.0, and *comparisons between two attributes*, for instance, Book.Title $<>$ Journal.Title. These constraints can be composed to complex integrity constraints by means of logical operators. The *extensional relationship editor* supports the derivation of extensional relationships from integrity constraints (e.g. Journal.ISBN<"1-2345" **and** Book.ISBN>"2-6789") and extensional assertions (e.g. Library.Book **disjoint to** Project.Book). The extensional relationships are expressed by the GIM-matrix which correlates disjoint extensions and classes of the participating schemata (see Section 5.1). Besides the automatic generation of the matrix using integrity constraints and extensional assertions, our tool also allows its direct manipulation.

The main part of **SIGMA**$_{Bench}$ is the *schema integrator* which automatically derives integrated object-oriented schemata using the source schemata and the modeled extensional relationships. This tool covers extensional and intensional conflicts and supports the integration of integrity constraints. The integration of access rights is currently in development. The *integration wizard* combines all necessary steps for the federated schema generation. It includes the *assertion generator* which currently supports attribute-constant-comparisons and specialization relationships. SpecTrac[1] allows to specify (global) transactions with different dependencies among them. The tool comprises a consistency checker which supports specifying consistent dependencies. In case of an inconsistency, alternative dependencies are proposed. Due to the fixed extensional assertions of the global schema, different commit rules can be derived for global transactions.

3 File Structure Analyzing

Traditional federated schema design approaches fail in integrating data files since files neither offer a data model nor a database schema which can be exploited for schema integration. It is important to mention that the integration has to cover all possible files of a given application scenario, and should not be restricted to only one or some example files [13]. In this context, we use the term *file cluster*

[1] A Tool for **Spe**cifying **C**onsistent **Tra**nsaction **C**losures

to refer to these file databases which cover all files of an application system. For the integration of such file clusters we suggest the derivation of a *structure description*, e.g. a grammar, describing the physical structure and the syntax of the data files and a *conceptual schema*, modeling structure, and semantics in database terms. The structure description could be a SGML-DTD (SGML Document Type Definition) or a context-free grammar according to YACC.

Completely structured files can be modeled with a grammar. Several languages for grammar definition exist, supported by tools like lexical scanners and parser generators. Such grammars can directly be compiled into adapters. A promising task is the semiautomated derivation of the grammar and schema. A top-down approach using grammatical inference methods is implemented in the NoDoSe-Tool [1]. Here, the developer defines the grammar by marking data in a text window and assigning them with data structures defined in a grammar or schema window. The tool calculates all candidates for tokens, identifying the data. In that way a structure description can be "mined" step-by-step.

Our approach as a part of the **SIGMA**$_{Bench}$ is a bottom-up approach. The first step is the syntactical analysis of a given set of files. The idea is to separate the words[2] in a file by delimiters. The set of delimiters can be defined explicitly. The next step is counting the occurrences for each word and sorting the list of words according to the count. Based on the hypothesis, that the most frequent words are good candidates for tokens or key words, this list can support the developer in grammar design. However, the experiments prove, that also non key words can occur very frequently and key words, e.g. identifying classes with very few instances in the files, can occur less frequently. Therefore, the developer has to check and to improve the list manually.

The next syntactical analysis step finds bracket pairs[3] or block delimiters. Based on a set of bracket pairs, the *SIGMA-File-Analyzer* constructs a parse tree for the files. The parse tree is translated into a grammar tree by a bottom-up approach beginning with the leaves of the tree. The goal is to find similar structured blocks in all levels of the tree. For each element in a block, two alternative representations are considered. The first one is the *pure* or immediate representation of a word, e.g., AUTHOR is represented as AUTHOR. This will mainly be used for key words or tokens. The second representation is a *meta*-representation by a type or class, e.g., the texts Gunter, Can, or Ingo can commonly be represented by the associated data type String. Checking all combinations of both representations should result in a reduced number of equivalent block patterns. The following example illustrates the simplification of inner block patterns. The parse tree contains the following inner blocks:

AUTHOR Gunter BOOK Databases
AUTHOR Can BOOK Objectbases
AUTHOR Ingo BOOK Schemaintegration
AUTHOR Michael BOOK Oracle8

[2] A word can also be a single character.
[3] Brackets can also be "long" words, e.g., BEGIN or END.

Regarding the number of four elements in the blocks, one can derive the complete set of 16 combinations. The first combination *(pure pure pure pure)* is the source combination. Setting the first element to meta does not reduce the number of blocks. The goal is to calculate the combination that has the fewest different blocks with as few as possible switches from *pure* to *meta*. In the example above, the best combination is *(pure meta pure meta)*:

AUTHOR String BOOK String

In this way, it is possible to reduce the number of different inner blocks. These blocks can now be represented by the calculated combinations.

This algorithm can recursively be applied to each level of the parse tree. The result is a grammar tree which is equivalent to the parse tree. The grammar tree can now be translated into a grammar for a adapter generator. The approach is partially implemented and produces good results for strongly structured files using blocks for structuring. Future work will enhance the technique to files where bracket-pairs are tokens with key word semantics or value semantics.

As argued in [18], a grammar is not a suitable approach for semistructured files. For instance, an HTML file, which is generated by a CGI script accessing a relational database, contains a huge volume of unnecessary and unstructured information in most cases. This information could be control sequences for text formatting, e.g., advertising of products, additional services, etc. Constructing a complete grammar for such a kind of file is both unnecessary and expensive. A more suitable approach for this kind of semistructured files is the application of pattern matching techniques, e.g., with regular expressions. This is done in the FLORID prototype [12] as well as in the WebJDBC system [18].

4 Assertion Generation

Integrity constraints are an important part of database schemata. They describe properties of the modeled real world and thus define the semantics of a database schema. Their enforcement ensures the semantic correctness of the corresponding database states. In order to build and maintain semantically correct database federations, locally existing integrity constraints must be reflected in the global schema. Furthermore, we may use local integrity constraints to derive global assertions for corresponding classes [27]. Extensional assertions are defined in order to relate classes of different schemata according to their extensional relationships. We denote the extension of the class C_1 in a database state S (the set of current instances) as $Ext^S_{C_1}$. Between the local classes the following extensional assertions [25] can be defined by the database integrator:

$$\begin{aligned}
\text{Disjointness: } & C_1 \varnothing C_2 :\Leftrightarrow \forall S\colon Ext^S_{C_1} \cap Ext^S_{C_2} = \emptyset \\
\text{Equivalence: } & C_1 \equiv C_2 :\Leftrightarrow \forall S\colon Ext^S_{C_1} = Ext^S_{C_2} \\
\text{Containment: } & C_1 \supset C_2 :\Leftrightarrow \forall S\colon Ext^S_{C_1} \supseteq Ext^S_{C_2} \\
\text{Overlap: } & C_1 \Cap C_2 :\Leftrightarrow \mathit{true}
\end{aligned}$$

Two classes are extensionally disjoint if there are no common objects in the extensions of these classes in any database state. Extensionally equivalent classes have same extensions in each database state. One class extensionally contains another class if the extension of the first class is a superset of the extension of the other class in any database state. The default value is overlap. In this case there are no restrictions on the extensions of the two related classes.

Integrity constraints can be exploited to give hints for extensional assertions between local classes [27]. In the following example, we show the derivation of extensional assertions for the local classes Library.Book and Project.Book. Suppose these classes consist of books of a library and a project, respectively. Furthermore, both of the local classes have an attribute Publisher with the following integrity constraints. In the library, there are only books published by Springer, Addison-Wesley, and Prentice-Hall. The books of the project are published by infix and Shaker:

Library.Book.Publisher $\in$ {Springer, Addison-Wesley, Prentice-Hall}
Project.Book.Publisher $\in$ {infix, Shaker}

Obviously, the extensions of classes Library.Book and Project.Book are disjoint. Moreover, the derivation of assertions based on linear arithmetic constraints is currently integrated into the tool *integration wizard*.

5 Integration

Integration concerns three parts, schema integration in general, integration of integrity constraints, and integration of access rights.

5.1 Schema Integration

Schema integration [3] is the main step in federated database design. In correspondence to the five-level-schema architecture [24] the schemata of existing component databases must be integrated to the federated schema. A complex task is to overcome schema heterogeneity. This step typically consists of detecting semantic conflicts and then of solving these conflicts by designing new schemata and corresponding schema mappings. After having solved all conflicts, the schemata can be merged into a federated schema.

In our project we developed the new schema integration method *GIM* [20, 21]. GIM is an acronym for *Generic Integration Model*, an intermediate data model. This model is designed for schema integration. The GIM approach enables the usage of algorithms from formal concept analysis [9] increasing the part of schema integration which can be done automatically.

In the following, the GIM-Method is sketched by an example. Assume, there is a library database storing information about publications, where books and journal papers are special types of publications. Each publication is either a book or a journal paper. This database has to be integrated with a project

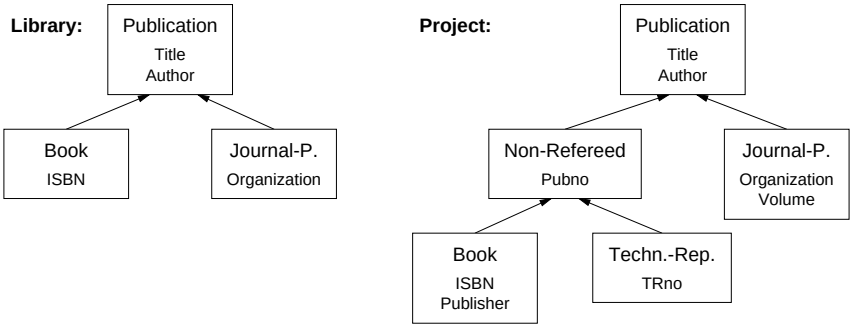

Fig. 3. Library and Project Schema

database that stores project publications. This database distinguishes more types of publications. Fig. 3 depicts the corresponding schemata of both databases.

Before these schemata are integrated, conflicts have to be resolved. In this paper we consider extensional and intensional conflicts. We assume that other types of conflicts are resolved as described, e.g., in [15, 25]. Extensional conflicts refer to redundancies among different classes whereas intensional conflicts are caused by different types of semantically related classes. The schemata define extensional subset relationships between sub- and superclasses and relationships due to integrity constraints. In general, it is impossible to completely conclude from given databases and integrity constraints whether the extensions of an arbitrary set of classes from different databases can simultaneously contain objects modeling a real-world object. Therefore, the designer has to give additional information about the extensional relationships among the classes.

Knowing the extensional overlaps the original extensions can be decomposed into disjoint *base extensions*. If, for example, the classes A and B overlap then their extensions are decomposed into three base extensions $(A \setminus B, B \setminus A, A \cap B)$. The algorithm presented in [22] computes the base extensions from the extensional assertions and is implemented in the $\mathbf{SIGMA}_{Bench}$.

The base extensions of our example are given in Fig. 4 (left). In our example, the original extensions are partitioned into base extensions 1–9. For example, the extension Book of the database Library is partitioned into the base extensions 1 and 9. The base extension 9 contains books from the library database which are simultaneously stored in the project database whereas the base extension 1 contains the remaining ones. Each publication, which is stored in the library or in the project database, can be assigned to exactly one base extension.

Now we have to compare the intensional aspects of the two schemata. For simplicity, we assume attributes with same names have identical semantics, i.e., *attribute conflicts* [15] are assumed to be resolved. The resolution of attribute conflicts is in general an important step during the integration process, but lies beyond the scope of this paper.

From the given extensional and intensional information we can automatically derive a table that assigns attributes to base extensions. A tick symbol for a

Local Classes	1	2	3	4	5	6	7	8	9
Library.Publication	✓	✓						✓	✓
Library.Book	✓								✓
Library.Journal-Paper		✓						✓	
Project.Publication			✓	✓	✓	✓	✓	✓	✓
Project.Non-Refereed			✓		✓	✓	✓		✓
Project.Journal-Paper				✓		✓		✓	
Project.Book					✓				✓
Project.Technical-Report			✓			✓			

Attr-Ext	1	2	3	4	5	6	7	8	9
Title	✓	✓	✓	✓	✓	✓	✓	✓	✓
Author	✓	✓	✓	✓	✓	✓	✓	✓	✓
ISBN	✓				✓				✓
Organization		✓		✓		✓		✓	
Volume				✓		✓		✓	
Publisher					✓				✓
TRno			✓			✓			
Pubno			✓		✓	✓	✓		✓

Fig. 4. Extensional Relationships and Extension-Attribute Relation

column (base extension) and a row (attribute) means that for potential objects of that base extension we know the value of that attribute (Fig. 4 right).

This table representation is a simplified GIM representation of the integrated schemata and can directly be used for further analyses. The order of columns and rows is irrelevant. By exchanging rows and columns we only obtain another representation. In such a representation a rectangle corresponds to a class: a union of base extensions having the same set of attributes. Moreover, we may detect subclass relationships using the rectangle representation of classes. If rectangles overlap, then they are in a specialization relationship. Algorithms of formal concept analysis [9] find all maximal rectangles and extensional subset relations between them. A maximal rectangle represents a class of the federated schema. The algorithm presented in [20], which is a slightly modification of the concept analysis algorithm, finds the following classes (maximal rectangles) from the Fig. 4 (right table):

C1 = ({1,2,3,4,5,6,7,8,9},{Title,Author})
C2 = ({4,6,8},{Title,Author,Organization,Volume})
C3 = ({5,9},{Title,Author,ISBN,Publisher,Pubno})
C4 = ({1,5,9},{Title,Author,ISBN})
C5 = ({6},{Title,Author,Organization,Volume,TRno,Pubno})
C6 = ({3,6},{Title,Author,TRno,Pubno})
C7 = ({3,5,6,7,9},{Title,Author,Pubno})
C8 = ({2,4,6,8},{Title,Author,Organization})

Now we build the matrix M which represents the irreflexive binary relation $<$. It expresses the specialization relation (subset relation) by comparing the extensions of each pair of classes. Two classes are in specialization relation if their sets of base extensions are in a subset relation. The computation $M_N = M \Leftrightarrow M \times M$ removes transitive specializations. Only the value '1' in a field of M_N represents a direct specialization relation between two classes.

The classes C1, ..., C8 and M_N can directly be used to produce the federated schema as depicted in Fig. 5. The names of the integrated classes correspond as far as possible to the classes of the two existing databases.

Besides the presented approach there are many other approaches to schema integration, e.g. [7, 17, 8]. The main difference between our approach and the

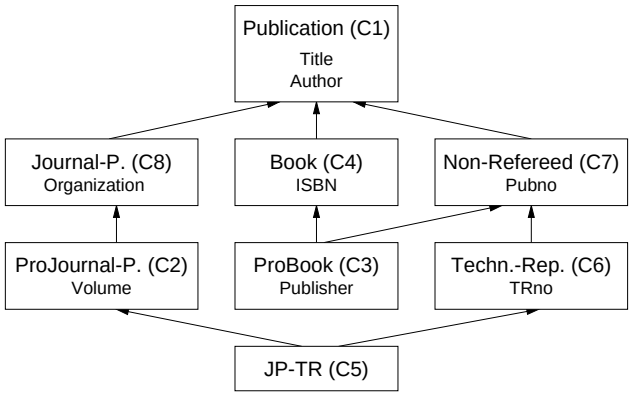

Fig. 5. Integrated Schema

other ones is the decomposition of extensions into base extensions and the application of efficient algorithms. We are able to integrate different inheritance hierarchies encompassing many classes. The other approaches explain how to integrate very few classes whereas our approach considers the schemata entirely.

5.2 Integrity Constraint Integrator

In order to build and maintain semantically correct database federations, locally existing integrity constraints must be reflected in the global schema. In this context, the notions of *global* and *local understandability* [28] play a central role. Global/local understandability means that a global/local transaction should not be rejected if it fulfills all integrity constraints of the corresponding schema. Our approach to dealing with integrity constraints during schema integration is based on a set of logical rules [6, 2]. Integrity constraints are usually given for a certain set of objects, i.e., they correspond to class extensions. Therefore, we always consider constraints together with the extensions for which they are valid. During integration we perform the following steps:

- Based on the GIM-Method, first the base extensions are derived from the given classes. Then, the respective integrity constraints are assigned to the base extensions. In doing so, we have to check which constraints of the original classes are still valid in the corresponding base extensions. Base extensions were created by extensional decomposition. Therefore, we call an integrity constraint *decomposable* if it remains valid on any subset of the original class extension [2]. Any constraint which is decomposable can be adopted unchanged to the corresponding base extensions. Please note that the decomposition of class constraints leads to a loss of information (which have to be compensated later during the composition phase). Obviously, object constraints are also decomposable, e.g., the constraint (price < 200.00) must hold also for the corresponding base extensions. Suppose that the extensions of the classes Library.Book and Project.Book overlap. Since the integrity

constraints of the class Library.Book are decomposable, they can be adopted to the base extensions 1 and 9. All decomposable constraints are combined conjunctively in the corresponding base extensions.
- When composing base extensions to global classes, we have to check whether the constraints are composable. An integrity constraint is called *composable* if it remains valid on any superset of the original extension. If a class is composed by the union of two base extensions which is not associated with any class constraints, then the corresponding object constraints of the base extensions can be disjunctively connected. However, base extensions usually contain class constraints such as uniqueness constraints. In this case, the problem is solved by introducing *discriminant*-attributes that are used to assign objects of the unified classes to the corresponding base extensions [6].

By integrating integrity constraints we can solve the global understandability problem. For solving the local understandability problem, the integration of integrity constraints must be done in a consistent way with respect to extensional assertions. However, since the extensional assertions are defined by the database integrator, the extensional assertions and the underlying local integrity constraints may be conflicting. Based on the idea that integrity constraints restrict the possible extensions of related classes, the derivation of extensional relationships can be partly supported as shown in [27] and Section 4.

5.3 Authorization Integration

A critical feature of database systems concerns the management of confidential data. Powerful mechanisms are needed for user management, authorization and authentication. In a federated environment, the global and local mechanisms have to work together to ensure global security. *Security* is achieved when the requirements *secrecy*, *integrity*, and *availability* are ensured.

Our focus lies on access rights. Access rights are statements about the rights of subjects (users, roles, programs, etc) to access objects (pieces of data) in certain ways (read, write, etc) including the right to grant them to other subjects [5]. In a federated system, local access rights have to be transformed, filtered and integrated. This includes the resolution of conflicts among overlapping access rights imported from different local systems.

During the design process of a federated system a global security model has to be chosen for expressing the integrated access rights. We express all access rights as triples consisting of the following concepts:

- *subjects*: only single users;
- *objects*: classes, objects, and attributes;
- *access types*: read, write, create, delete.

We have only positive access rights and we follow a closed world policy. This model can be extended with more complex concepts like user role models.

Fig. 6 illustrates a scenario with two local databases Project and Library. The local user named joe has read access to the class Project.Publication over the role

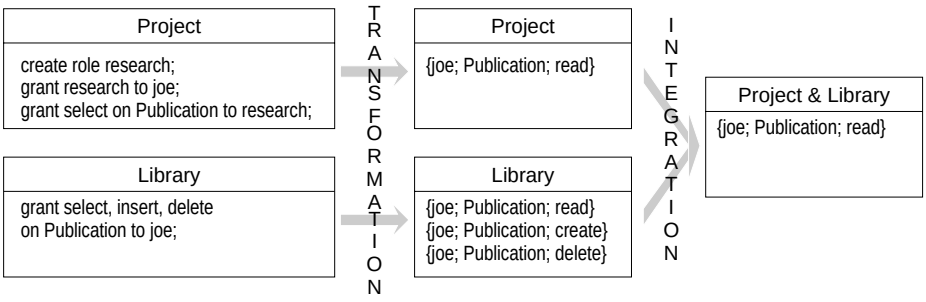

Fig. 6. Transformation and Integration of Access Rights

research. In the other system, the local user joe may read and write the class Library.Publication.

At first, the access rights on the local schemata are transformed into access rights expressed in terms of our global model. Because of the simplicity of our model, we have to divide access rights based on complex concepts like for example groups of users or views which in turn increases the number of access rights significantly. In our example this affects the role research to which joe is assigned.

Afterwards, the exported local access rights are integrated into global access rights which correspond to the integrated schema. A semantic conflict may arise if a user has the right to access a certain data object in one local system but not the same or overlapping data object in another local system. Consider again Fig. 6. During schema integration the local classes Project.Publication and Library.Publication are integrated to the global class Publication (cf. Fig. 5). We integrate the users from the different local systems with a special user management tool that imports the local user profiles and ensures that all identical users are assigned with one global identity. In the case of our example we create the global subject joe which is joined with its two local counterparts. So, a conflict arises because joe has only read access to the corresponding local class Project.Publication and complete access rights to the other local class.

The designer of the federated system has the following three possibilities to resolve such a conflict in cooperation with the local administrators:

- The access is denied if this is required by the relevant administrator.
- The access is permitted if this is accepted by the relevant administrator.
- The right is locally granted. The conflict vanishes.

In our example we choose the first option. Thus, the user joe is only allowed to update Library.Publication at the local system.

Finally, we have to discuss how we use the integrated access rights within the scope of a federated authorization process. In the tool **SIGMA**$_{Bench}$ we implement a variant of global authorization that checks all global queries concerning federated data against the integrated access rights. At the local systems, we establish a user with the login name fdbs to whom the responsible administrator grants full access to all data that can be exported. To obtain the necessary

trust of the local systems for this approach, we additionally need powerful user authentication mechanisms at the global side [11].

6 Global Transactions

In a federated database environment, *global transactions* are able to transparently access and manipulate data located in different local database systems [4]. Depending on the underlying global schema, and particularly on the fixed extensional assertions, a global transaction is decomposed into a set of *global subtransactions* which operate on the local database systems. Extensional assertions influence the dependencies between the global subtransactions of the same global transaction [29]. Different *global commit rules* are needed to correctly terminate a global transaction. For instance, in some cases a commit of only one global subtransaction is sufficient to commit the global transaction, i.e., a global transaction can commit although some of the global subtransactions abort.

In the following, we consider the decomposition of a global transaction t_g and the corresponding global commit rule for this transaction. The global transaction t_g consists of the following insert on the global classes C_{g_1} and C_{g_2} (where the classes C_1 and C_3 are managed in one local database system while the classes C_2 and C_4 are managed in another local database system):

Insert into $C_{g_1} = C_1 \cup C_2$ with $C_1 \oslash C_2$;
Insert into $C_{g_2} = C_3 \cup C_4$ with $C_3 \supset C_4$;

The insertion into class C_{g_1} is successful if the corresponding object is inserted in exactly one of the local classes C_1 and C_2. In contrast, an object is inserted into class C_{g_2} if the object appears at least in class C_3. The global transaction is successful if either an object is inserted in class C_1 and class C_3, in class C_2 and class C_3, in the classes C_1, C_3, and C_4, or in the classes C_2, C_3, and C_4.

If only one of the operations of a transaction aborts then the whole transaction aborts. Thus, the global transaction is decomposed into global subtransactions such that transaction t_i performs the insertion into class C_3, transaction t_j into class C_1, transaction t_k into class C_2 and transaction t_m into C_4. The resulting dependencies are as follows (cf. termination dependencies in [23]):

- The abortion of transaction t_g requires the abortion of all subtransactions: $a_{t_g} \Rightarrow (a_{t_i} \wedge a_{t_j} \wedge a_{t_k} \wedge a_{t_m})$.
- The commit of transaction t_i is essential for transaction t_g: $(a_{t_i} \Rightarrow a_{t_g})$.
- If the transactions t_j and t_k abort, then the global transaction t_g has to abort, too: $(a_{t_j} \wedge a_{t_k}) \Rightarrow a_{t_g}$.
- Transaction t_j and t_k are in an exclusive dependency: $(c_{t_j} \Rightarrow a_{t_k} \wedge c_{t_k} \Rightarrow a_{t_j})$.
- The success of transaction t_m is optional.

The previous example shows that the assumption in the literature that at least one global subtransaction of the same global transaction is executed at the corresponding local systems, e.g., [4, 10], cannot hold since local integrity constraints are not known and considered on the global level. The tool SpecTraC enables us to specify arbitrary sets of transactions on the global level. These are connected over transaction dependencies like termination dependencies.

7 Conclusions

Building and maintaining federated database systems is a hard task which cannot completely be done in an automatic way. Our main goal is to support the schema designer of a federated database during the design process. We work on different areas of this complex problem and develop a bunch of solutions. The online documentation as well as a more detailed technical report of the **SIGMA**$_{Bench}$ can be found under:

http://wwwiti.cs.uni-magdeburg.de/~sigmafdb/

From a scientific point of view, there is still a lot of open problems in database integration. One main problem is the extraction of semantic information from existing systems. Our approach has done first steps in analyzing file structures, database schemata, and existing explicit integrity constraints. However, schema re-engineering and extraction of implicit integrity constraints can never be completely automated and, thus, always require human interaction.

Besides the concrete federation aspects shortly described in this paper there is still a large number of other facets where methods and tool support are needed. Examples are integration of stored procedures, other databases models, general behavior integration, etc. Our aim is to build an open tool environment where developments in these areas can be 'plugged in' to enrich the design support offered by our tool.

References

1. A. Adelberg. NoDoSE - A Tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents. In L. Haas, A. Tiwary, eds., *Proc. SIGMOD'98*, ACM SIGMOD Record, Vol. 25, pp. 283–294, ACM Press, 1998.
2. S. Balko, C. Türker. Integration of Aggregation Constraints. In S. Conrad, W. Hasselbring, G. Saake, eds., *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS'99*, pp. 5–24. infix-Verlag, 1999.
3. C. Batini, M. Lenzerini, S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
4. Y. Breitbart, H. Garcia-Molina, A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 1(2):181–240, 1992.
5. S. Castano, M. G. Fugini, G. Martella, P. Samarati. *Database Security.* ACM Press, Addison-Wesley, 1995.
6. S. Conrad, I. Schmitt, C. Türker. Considering Integrity Constraints During Federated Database Design. In S. M. Embury, N. J. Fiddian, A. W. Gray, A. C. Jones, eds., *Advances in Databases, BNCOD 16*, LNCS 1405, pp. 119–133. Springer, 1998.
7. U. Dayal, H. Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Transactions on Software Engineering*, 10(6):628–644, 1984.
8. Y. Dupont, S. Spaccapietra. Schema Integration Engineering in Cooperative Databases Systems. In K. Yetongnon, S. Hariri, eds., *Proc. 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems, PDCS'96*, pp. 759–765. International Society for Computers and Their Application, 1996.
9. B. Ganter, R. Wille. *Formal Concept Analysis.* Springer, 1998.

10. D. Georgakopoulos, M. Rusinkiewicz, A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):166–180, 1994.
11. E. Hildebrandt, G. Saake. User Authentication in Multidatabase Systems. In R. Wagner, ed., *Proc. 9th DEXA Workshop*, pp. 281–286. IEEE CS Press, 1998.
12. R. Himmelröder, B. Ludäscher. Querying the Web with FLORID. In M. H. Scholl, H. Riedel, T. Grust, and D. Gluche, eds., *10. Workshop "Grundlagen von Datenbanken"*, Forschungsbericht No. 63, pp. 94–98, Universität Konstanz, Fachbereich Informatik, 1998.
13. M. Höding. An Approach to Integration of File Based Systems into Database Federations. In *Heterogeneous Information Management, Proc. 10th ERCIM Database Research Group Workshop*, pp. 61–71, ERCIM-96-W003, European Research Consortium for Informatics and Mathematics, 1996.
14. M. Höding, R. Hofestädt, G. Saake, U. Scholz. Schema Derivation for WWW Information Sources and their Integration with Databases in Bioinformatics. In [16], pp. 296–304.
15. J. A. Larson, S. B. Navathe, R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989.
16. W. Litwin, T. Morzy, G. Vossen, eds., *Proc. ADBIS'98*, LNCS 1475, Springer, 1998.
17. A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13(7):785–798, 1987.
18. K.-U. Sattler, M. Höding. Adapter Generation for Extraction and Querying Data from Web Sources. In *Proc. 2nd ACM SIGMOD Workshop WebDB'99*, 1999.
19. I. Schmitt, G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In B. Thalheim, ed., *Conceptual Modelling — ER'96*, LNCS 1157, pp. 195–210. Springer, 1996.
20. I. Schmitt, G. Saake. Merging Inheritance Hierarchies for Database Integration. In M. Halper, ed., *Proc. CoopIS'98*, pp. 322–331. IEEE CS Press, 1998.
21. I. Schmitt, G. Saake. Integrating Schemata using the GIM Method. Informatik Preprint, Universität Magdeburg, 1999.
22. I. Schmitt, C. Türker. Refining Extensional Relationships and Existence Requirements for Incremental Schema Integration. In G. Gardarin, J. French, N. Pissinou, K. Makki, L. Bougamin, eds., *Proc. CIKM'98*, pp. 322–330. ACM Press, 1998.
23. K. Schwarz, C. Türker, G. Saake. Extending Transaction Closures by N-ary Termination Dependencies. In [16], pp. 131–142.
24. A. P. Sheth, J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
25. S. Spaccapietra, C. Parent, Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, 1992.
26. K. Schwarz, I. Schmitt, C. Türker, M. Höding, E. Hildebrandt, S. Balko, S. Conrad, G. Saake. Tool Support for the Design of Database Federations in SIGMA$_{FDB}$. Informatik Preprint 14, Universität Magdeburg, 1999.
27. C. Türker, G. Saake. Deriving Relationships between Integrity Constraints for Schema Comparison. In [16], pp. 188–199.
28. C. Türker, G. Saake. Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration. In *Proc. ADBIS'99*, LNCS. Springer, 1999.
29. C. Türker, K. Schwarz, G. Saake. Commit Protocols for Global Transactions in Federated Database Systems. Informatik Preprint 2, Universität Magdeburg, 1999.