Combining a Formal with an Example-driven Approach for Data Integration

Ingolf Geist Kai-Uwe Sattler Ingo Schmitt
Department of Computer Science
University of Magdeburg
P.O.Box 4120, 39016 Magdeburg, Germany
{geist|kus|schmitt}@iti.cs.uni-magdeburg.de

Abstract

Integrating data sources is a general problem in many scenarios. The main problem is the heterogeneity between data sources which were created and developed separately. In the literature there exist many different approaches to solve that problem. Schema integration approaches derive an integrated schema by resolving conflicts on schema and data model level. Another kind of approaches are multidatabase languages and systems which define and verify views on top of example databases. They try to map local data to a predefined, integrated schema. In this paper we investigate how the schema integration approach GIM and the example-driven approach VIBE basing on FRAQL can be combined for a new combined approach. The combined approach benefits from the powerful mapping language FRAQL and the verification feature by using example databases and from the extensional analysis of the GIM approach. We discuss the advantages of such a combined approach and show its potential in many integration scenarios.

1 Introduction

The problem of integrating data from heterogeneous sources is addressed by various approaches developed in the previous years. However, in most cases these approaches assume a scenario, where a new global and integrated schema has to be derived and instance-level conflicts are considered afterwards or separately. Recently, integration approaches where proposed which focus on alternative or complementary paradigms, e.g. a yo-yo-like approach or example- and data-driven techniques.

Considering more complex integration tasks in real-world scenarios with (partially) given global schemas, e.g. based on domain reference models, an approach combining formal schema integration with example-driven analysis and conflict resolution seems to be most promising. Based on our previous work in the field of data integration we present in this paper our ideas on such a combined approach.

The remainder of this paper is organized as follows. In Section 2 we introduce our previously developed integration techniques. Based on this, we discuss a combined method addressing the definition of mappings between global and local schemas as well as the resolution of schema-level and instance-level conflicts in Section 3. Section 4 gives an overview of available tools and sketches their usage as part of an integrated tool set for the combined method. Finally, in Section 5 we discuss related work and conclude the paper in Section 6.

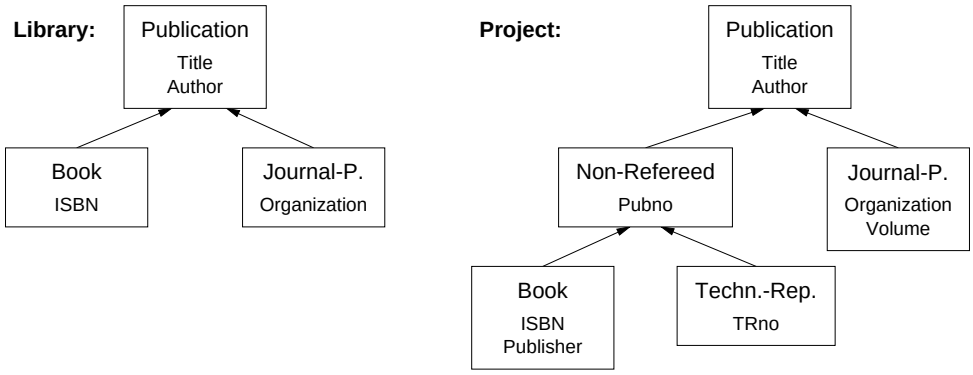

Figure 1: Library and Project Schema

2 Formal vs. Example-driven Integration

2.1 GIM Method

Schema integration [BLN86] is the process of generating one homogeneous database schema from several, heterogeneous source schemas. Besides the integrated schema the integration produces appropriate schema mappings. 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 schemas and corresponding schema mappings. After having solved all conflicts, the schemas can be merged into an integrated schema. In our project **SIGMA**$_{FDB}$ we developed the schema integration method *GIM* [SS98, SS99, Sch98]. GIM is an acronym for *Generic Integration Model* – an intermediate data model and stands simulataneously for a specific integration method. This model is designed for schema integration. The GIM approach enables the usage of algorithms from formal concept analysis [GW98] increasing the portion of schema integration which can be performed 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 database that stores project publications. This database distinguishes more types of publications. Fig. 1 depicts the corresponding schemas of both databases.

Before these schemas are integrated, conflicts have to be resolved. Here we consider extensional and intensional conflicts only. We assume that other types of conflicts are resolved before as described, e.g., in [LNE89, SPD92]. Extensional conflicts refer to potential object redundancies among different classes whereas intensional conflicts are caused by different object types of semantically related classes. The schemas 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 same real-world objects. Therefore, the designer has to give additional information about the extensional relationships among the classes.

Knowing the potential extensional overlaps the original potential extensions can be decomposed into disjoint *base extensions*. If, for example, the classes A and B extensionally overlap then their extensions are decomposed into three base extensions ($\mathsf{A} \setminus \mathsf{B}, \mathsf{B} \setminus \mathsf{A}, \mathsf{A} \cap \mathsf{B}$). The algorithm presented in [ST98] computes the base extensions from extensional assertions and is implemented in the **SIGMA**$_{Bench}$.

The base extensions of our example are given in Fig. 2. In our example, the original exten-

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			✓			✓			

Figure 2: Extensional Relationships

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

Figure 3: Extension-Attribute Relation

sions 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 (or in both), can be assigned to exactly one base extension.

Now we have to compare the intensional aspects of the two schemas. For simplicity, we assume attributes with same names have identical semantics, i.e., *attribute conflicts* [LNE89] are assumed to be resolved before.

From the given extensional and intensional information we can automatically derive a table that assigns attributes to base extensions. A tick symbol for a column (base extension) and a row (attribute) means that for potential objects of that base extension we know the value of that attribute (Fig. 3).

This table representation is a simplified GIM representation of the integrated schemas and can be directly used for further analyses. The order of columns and rows is irrelevant. By exchanging rows and columns we only obtain other representations. 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 [GW98] find all maximal rectangles and extensional subset relations among them. A maximal rectangle represents a class of the integrated schema. The algorithm presented in [SS98], which is a slight modification of the concept analysis algorithm, finds the following classes (maximal rectangles) from Fig. 3:

C1 = ({1,2,3,4,5,6,7,8,9},{Title,Author})
C2 = ({4,6,8},{Title,Author,Organization,Volume})

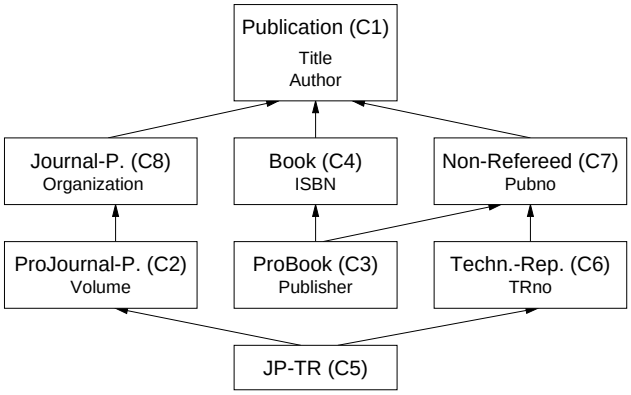

Figure 4: Integrated Schema

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 $M_N = M - M \times M$ removes transitive specializations. The classes C1, ..., C8 and M_N can be directly used to produce the integrated schema as depicted in Fig. 4. The names of the integrated classes correspond as near as possible to the classes of the two existing databases.

2.2 Example-driven Integration

After the description of the integration method using GIM we want to discuss a second integration technique: the *Example-driven Integration*. This approach is a supplement to the GIM method and assumes, that a preliminary integrated schema already exists. Such an integrated schema can be created by global requirements or by using a domain reference model. Given the integrated schema, example-driven integration is the process of creating iteratively and interactively mappings between local schemas and the global schema driven by evaluation of the current data. During this process integration conflicts between global and local schemas have to be resolved which can lead to a refinement of the integrated schema. An integration step consists of two phases – *conflict resolution* and *analysis* – which are performed iteratively.

The input of the example-driven approach is formed by the global schema, the local schemata and the local data. Further, interaction with the user is required. The output is a mapping between the local sources and the global schema as well as the refinement of the global schema. Thus, a technology is needed that is powerful enough to resolve conflicts and that permit a unified access to the local data. *Multidatabase systems* and their query languages (e.g. [GLRS93, TAH+96, LSS96]) form a basis for example-driven integration. We will utilize the language FRAQL[1] [SCS00] for our purposes.

[1] **F**eder**a**ted **Q**uery **L**anguage

This query language resolves schema and data conflicts by creating view definitions. There exists two different kinds of views: import and integration relations. These definitions allow the resolution of following integration conflicts: attribute conflicts, structure conflicts, meta conflicts, extensional conflicts (only for two sources) as well as same-object conflicts and data value conflicts. For details of the conflict resolution features see [SCS00]. Within the view definitions mapping functions and tables are used resolving same-object conflicts and data conflicts. These functionalities support the first part of the integration – the conflict resolution. The example-driven integration approach can be described in the environment of multi-database systems as an interactive and successive definition of integrating views.

Since the integration conflict resolution is a complex task an iterative process has to be considered. Therefore the single steps are created using a approach inspired by QbE[2] [Zlo77]. A table view is used to specify the integration task by giving examples and showing the integration results immediately. Thereby the special conflict resolution operation of FRAQL are taken into account. Additionally an iconic view of the integration graph is given which forms the view definition. This definition can be changed intuitively.

After the description of the first phase we want to discuss the second phase: the evaluation of the created – possible intermediate – results of the view execution. We use the facilities of FRAQL to execute the created views. So, the access to the resulting data is possible. Different analysis methods have to be used for the various conflict resolution steps. They can be simple queries or also more complex functions. The results of the analysis are interpreted by the user. Based on these interpretations the integrator refines the integrated schema.

Conflicts occur in different ways during the integration steps. The analysis phase tries to identify these conflicts. The occurrence of *null values* and *null columns*[3] can indicate that structure conflicts are not handled in the global schema. One reason can be that one or both sources do not have enough information to fill out the global schema. Therefore the integrated schema has to be refined or additional sources have to be added.

A second point is the evaluation of correspondence assertions, that means, the examination of extensional relationships between the classes. The first step is using cardinalities as a heuristic function for discovering possible problems according to same-object conflicts or wrong assumptions about the assertions. Fig. 5 shows the expected cardinalities for results of the union operation. If there is a conflict, the user can apply set oriented operations to get more

assertion	**expected cardinality:** $\mathbf{card(R)}$
$R_1 \equiv R_2$	$card(R) = card(R_1) = card(R_2)$
$R_1 \subseteq R_2$	$card(R) = card(R_2)$
$R_1 \cap R_2$	$\max(card(R_1), card(R_2)) \leq card(R) \leq card(R_1) + card(R_2)$
$R_1 \not\equiv R_2$	$card(R) = card(R_1) + card(R_2)$

Figure 5: Cardinality heuristics: $R = R_1 \cup R_2$

detailed results about the nature of the conflict. But the evaluation of extensional relationships requires a previous solution of the same-object conflict.

The verification of a same-object conflict solution is another application area of data oriented integration. A created mapping table or function can be evaluated on the data. Thereby data mining techniques are useful.

[2] **Q**uery **b**y **E**xample

[3] All values of the column are null.

The attribute value conflicts can be discovered by following steps. The first step is the computation of an outer join of the local relations. Second, we can compare the corresponding values by using a user-defined function. There are two cases of different values. The first case indicates that the supposed mapping is not correct. For instance, an attribute conflict is not resolved, and the mapping and/or the integrated schema have to be modified. The second reason of data conflicts are data errors or different update times of the local systems. Data errors can only be handled in the local data sources, but an examination of an example set of error pairs can provide an idea about which local system is more reliable. Thereupon a mapping function can select in a conflict case the value from the more trustworthy source.

3 An Iterative Method for Defining Schema and Instance Mappings

In the previous section we introduced the schema integration method GIM and the example-driven approach. They are used in different scenarios. Whereas the schema integration creates an integrated schema, it is already given for the example-driven approach. Different to the schema integration the example-driven approach creates views on example databases which allow an interactive definition of views and data mappings. Both approaches, however, have to overcome conflicts caused by schema and data heterogeneity. In the following we will examine how both approaches can be combined for different scenarios.

3.1 Scenario discussion

There are many different scenarios where the combination of the GIM and the example-driven approaches makes sense. We can at first separate scenarios with predefined integrated schema from scenarios without predefined integrated schema.

Scenario without given integrated schema

This is the classical schema integration scenario. The integrated schema is produced by the schema integration process. The example-driven approach can be used to check the integrated schema and the schema mappings against the current databases states. The check can be performed after the integration process has been completed as well as after every single integration step. The result can be used for refinement.

Scenario with given integrated schema

This scenario is the more interesting scenario. In many situations we already have an integrated schema. Due to the integrated schema we have in addition to horizontal schema conflicts vertical conflicts as depicted in Fig. 6.

Schema integration is designed to overcome horizontal conflicts and the example-driven approach to solve vertical conflicts. The question arises whether it is better to resolve horizontal first or vertical conflicts first. If we start to reconcile horizontal conflicts there is a good chance that we do not move forward to the integrated schema, that is, many vertical conflicts have to be resolved afterwards. If we build a schema by schema integration there is a high probability of vertical conflicts between it and the given integrated schema. The problem is caused by the fact that the schema integration does not consider the given integrated schema. One way to avoid this problem would be to regard the integrated schema as the third local schema. After the schema

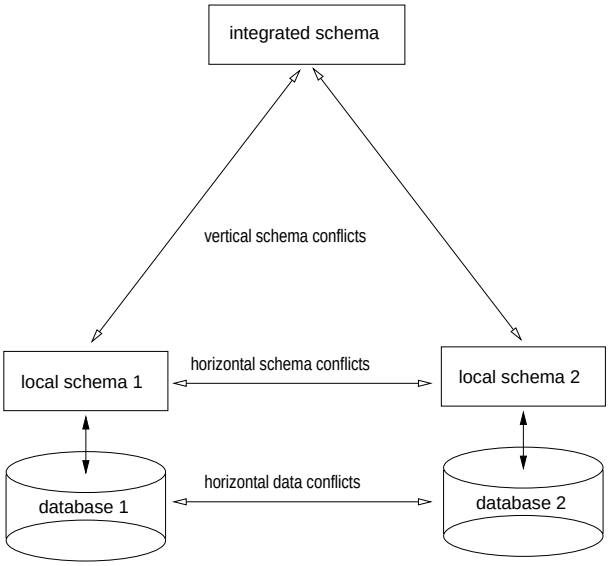

Figure 6: Horizontal and vertical conflicts

integration produced a new schema all mappings between the original local schemas and the original integrated schema are defined over the new integrated schema. Since the integrated schema is considered as a local schema there can occur many unnecessary mappings.

A better approach is to resolve vertical conflicts first since resolving vertical conflicts resolves most horizontal conflicts implicitly (on resulting intermediate schema levels). But it is not as easy as it looks like. Vertical conflict resolution means to overcome the conflicts between the integrated schema and one local schema. But not all conflicts can be regarded as purely vertical conflicts. In the following we list the different kinds of conflicts which can be resolved without knowledge of the parallel local schema:

- structure conflict: An attribute in one schema is a class in the other schema.
- attribute conflict: There are different attribute value representations. One attribute can encompass more information than another one. Sometimes, mapping functions are required to bridge different attribute values semantically.
- meta conflict: One attribute value in one schema can correspond to a class in another schema in the sense that all objects have implicitly the same meta value.
- intensional conflict: Equivalent classes of the integrated schema and a local schema can have different sets of attributes.

The following remaining conflicts cannot be resolved in a purely vertical manner:

- extensional conflict: The extensions of local classes stand in specific relationships. The goal in this scenario is to define mappings which relate local extensions to the integrated schema. But at the same time horizontal, extensional overlappings must be considered since they express redundancy to be dealt with. Defining extensional mappings in consideration of horizontal extensional conflicts can cause intensional conflicts: for some objects there are no values for the required attributes of the corresponding class[4].

[4]Since global updates are not supported in our scenario we ignore conflicting integrity constraints here.

- same-object detection: Database objects from different local databases referring to same real-world objects need to be detected.

- data conflict: In case of redundant data between the local databases there can exist differences of attribute values although they should be same. This occurs, for example, if one local database contains newer data than the other one.

In the following subsection we will discuss the resolution of vertical conflicts. In Section 3.3 we focus then on the non-vertical conflicts.

3.2 Views for Resolving Vertical Conflicts

The first step in our proposed method is the resolution of conflicts between the global schema and each local schema separately. This is usually supported in multidatabase languages and federated database systems by defining views on local relations. In the following we discuss the resolution of vertical conflicts using the FRAQL language. However, most examples presented here could be translated to other languages or systems, e.g. IBM DataJoiner [VZ98], SchemaSQL [LSS96] or Pegasus [ASD+91].

FRAQL provides a view mechanism that distinguish between two kind of views: import and integration views.

An *import view* is a projection of a local relation of a data source. The import view is defined by specifying the source relation and, if required, a mapping between local and global attributes.

```
create view global_name of type_name
  as import from source.local_name
  [ mapping_definitions ];
```

In the view definition given above, the attribute mapping can be described in the following variants:

- Without an explicit mapping, a local attribute corresponding to an attribute, defined by the global object type, in terms of identifier and type becomes an attribute of the global relation.

- The notation *g_name* **is** *l_name* means renaming the local attribute to *g_name*. This requires type compatibility.

- The notation *g_name* **is** *func(l_name)* defines that the global attribute value is calculated by using the user-defined conversion function *func* on the local attribute value.

- The definition *g_name* **is** *@tbl (l_name, src, dest, default)* means that the database table *tbl* is used for mapping the values from the local attribute *l_name*. The value of the global attribute is obtained by looking for the value of attribute *l_name* in column *src* and retrieving the corresponding value of column *dest*. The field *default* denotes a default value, either as literal or as local attribute, which is assigned to the global attribute, if the value of *l_name* is not found in the table. In fact, this kind of attribute mapping is evaluated by a left outer join, where the `NULL` value is replaced by the default value *default*.

The following example illustrates the usage of these mapping concepts: Given a source relation Book (isbn, title, category, retailer, price) an import could be defined, where the local attribute retailer is mapped to dealer, the values of attribute category are mapped via a table cat_map and the prices are converted into Euro prices:

```
create view Books of BookType as import from src.Book (
    dealer is retailer,
    category is @cat_map (category, cat_src, cat_dest, NULL),
    price is dollar2euro (price)
);
```

In this way, features like mapping functions or tables are particular helpful for resolving structure, attribute and intensional conflicts.

The second kind of views is called *integration view* and extends ordinary SQL views by features like meta-data access and variable substitution. Variables of a query can not only be bound to relations as tuple variables, but also to meta-data, like the set of attributes of a relation or the set of relations of a schema. This was originally proposed in SchemaSQL [LSS96]. But in contrast to SchemaSQL, where meta-data access in queries is implemented as a language extension, in our approach the schema catalog is used. So, the catalog relation `catalog.columns` contains information about attributes of all global relations, whereas the relation `catalog.tables` describes the global relations. Unlikely SchemaSQL, any global user relation with information about other relations can be used as meta-data source.

As an extension to standard SQL, attributes of tuple variables in queries can be obtained during evaluation. This means, while in SQL names of attributes and relations are constants, in FRAQL they can be constructed from current values of other tuple attributes. This *variable substitution* is written in the notation `$var` and can appear everywhere in a query, where names of attributes or relations are expected. For example, the expression `tbl1.$(tbl2.col)` means the attribute value of the current tuple of relation `tbl1`, whose name is obtained from the current value of `tbl2.col`.

Variable substitution and meta-data access are useful for resolving meta conflicts. Assuming a global relation Book (isbn, title, category, dealer, price) and a local relation as shown in Fig. 7 the local relation can be restructured by the following view definition:

```
create view BookPrices of BookType as
  select b.isbn, b.title, b.category, b.$(c.column_name),
    c.column_name
  from BookStore b, catalog.columns c
  where c.table_name = 'BookStore' and
    c.column_name = 'dealer1' or c.column_name = 'dealer2';
```

isbn	*title*	*category*	*dealer1*	*dealer2*
123-XX	Database Systems	Computer Science	19.90	21.90
234-YY	Data Warehouses	Computer Science	43.50	40.00

Figure 7: Meta conflict in table BookStore

Ideally, based on these views the further steps of the integration process have only to consider the remaining kind of conflicts, particularly extensional conflicts.

Classes	1	2	3	4	5	6
Books			✓	✓	✓	
Book	✓	✓	✓	✓	✓	
Textbook				✓	✓	
Buch			✓	✓		

Figure 8: Extensional Relationships

3.3 Extensional conflicts

The GIM approach as well as the example-driven approach deal with extensional conflicts. In the example-driven approach the binary extensional relationships are considered only. The GIM approach, however, is not restricted to binary relationships but suffers sometimes from hard to understandable extension tables. The problem with binary, extensional comparisons occur if more than two classes are extensionally involved. This is possible even if only two local databases are considered due to potential specializations between sub- and superclasses. It can be easily shown that expressing extensional relationships among three classes cannot sufficiently expressed by three binary assertions. Due to this argument we adopt the GIM approach for dealing with extensional conflicts. We use the example-driven approach to check specified extensional relationships.

As already described in Section 2.1 the potential extensions of the local classes must be related to each other in an extension table. But what about the classes of the given integrated schema? They do not have real extensions. But its designer has specific potential extensions in her/his mind. If he names a class `Person` than he obviously does not expect to get books. For the extensional analysis it is therefore important to relate the potential extensions of the integrated schema classes to the extensions of the local classes. This has to be done in the one extension table.

Consider the following example: One local schema contains the class `Book` and a subclass `Textbook` whereas the second local schema includes the class `Buch`. In the integrated schema we have a class `Books`. In Fig. 8 we relate the four classes to each other by using background knowledge about the potential local extensions and the intended extension of class `Books`.

The problem is now to map the local extensions to the global extensions. We distinguish four different cases:

1. set operation: The extension of a global class can be expressed by applying set operations to the local class extensions. An interesting question is here to find an optimal set expression w.r.t. costs for different set operations.

   In our example we can define: `Books` := `Textbook` ∪ `Buch`.

2. selection: Unfortunately, it is not always possible to express a global extension by a set expression. Sometimes we can additionally use a select operation since a local attribute helps us to select a right subset of a local extension.

   If in our example the class `Books` has the intended extension of 2-5, then there is no way to distinguish base extension 1 from 2 by set operations. But we can use an attribute of class `Book` to separate both base extensions. For example, the intended extension of `Books` should not include poem books. If there is an attribute `type` of class `Book` then we can use it to exclude poem books.

3. gap: In this case the intended extension of a global class cannot be completely filled by

local potential extensions. There is no remedy for this case. The designer must decide, if he changes the intended extension or even the integrated schema. He can also argue, that the missing objects can come later from a third local database.

In our example we have a gap if the intended extension of class `Books` includes base extension 6.

4. else case: Here we have neither a gap nor a set expression is possible. Furthermore, no selection can be found to generate the intended extension. This severe problem cannot be solved by mapping. Similar to the previous case, the designer has to make the choice between a change of the intended extension or a modification of the integrated schema.

   We have this situation if `Books` should not include poem books but there is no way to distinguish poem books from other book types.

From the example-driven approach we can now adopt the extension check. We assume that we already know how to detect same-objects. The extension check must be adopted to our extension table. A correct extensional table means that every local object can be assigned to exactly one base extension (one column). There are two kinds of error. First, for one local object there is no corresponding base extension, e.g. an object of `Buch` being not in `Book`. Second, some base extensions are always empty, e.g. because there is no overlap as specified. The example databases can now be used to detect those errors. Unfortunately, only the errors of the first kind can be found. If a local object does not have a base extension then we have to create a new base extension. However, if all local objects of the example databases can be correctly assigned to their corresponding base extensions then we do not know whether this is still the case when the local databases are updated.

After the extensional analysis has been finished we have to consider intensional conflicts. We must check if for the extension of every global class all values for the specified attribute can be derived. This can only be done completely if the extensional analysis has been finished. For this check we create a GIM diagram which assigns attributes to base extensions. A tick in a cell means that we have for every potential object of that base extension the value for the corresponding attribute. Every global class can be regarded as a rectangle in the diagram. For a correct mapping we require that every rectangle representing a global class contains no cell without a tick.

If a rectangle contains empty cells then we have a conflict. A simple solution is to fill this cell by generated null or default values. This solution is, however, only a small-scale solution. Otherwise it would enable the generation of a universal table with too many null or default values. The alternative to null values is again a modification of the integrated schema. As modification the designer can drop an attribute or a base extension from the global class. If there are too many conflicts and a modification is too complex then the designer can apply the GIM algorithm to generate an new, integrated schema by using mechanisms of formal concept analysis.

In addition to detecting extensional and intensional conflicts the GIM diagram can be used to check specializations extensionally between global classes. A correct specialization means the a subclass extension is always a subset of the superset extension. Since the integrated schema has been designed at a time when the extensional mappings to the local classes were unknown this kind of conflict can easily occur. If such a conflict happen then again the mappings or the global classes must be modified.

Last but not least, if the extensional and the intensional analysis satisfies the requirement of the integrated schema we have to deal with data conflicts. Data conflict means different attribute

values from different local databases for the same attribute and base extension. The designer has to specify how global values can be derived from the local values.

From the result of the described steps we are now able to create the final mappings expressed in FraQL.

4 Tool Support

Due to the complexity of the integration task, one-shoot strategies are often not realistic, particularly for larger projects. Thus, integration has to be considered as an iterative and interactive process, that requires tool support enabling the definition and evaluation of global, integrated views and mappings to the local schemas. This includes features like deriving and modifying an integrated schema automatically as well as providing hints about potential conflicts and applying resolution functions in an interactively manner. In the following sections we present first the already available tools supporting both integration paradigms separatly. Finally, we sketch how these tool sets could be combined in order to meet demands of real-world integration scenarios.

4.1 VIBE - Visual Integration by Example

A prototype called VIBE[5] was developed to support the example-driven approach. VIBE is developed in Java and accesses the FRAQL via the standard database API JDBC. FRAQL is utilized for performing queries over heterogeneous data sources and defining the integrated schema. The catalog of FRAQL is used to store the integration information. The architecture is shown in Fig. 9.

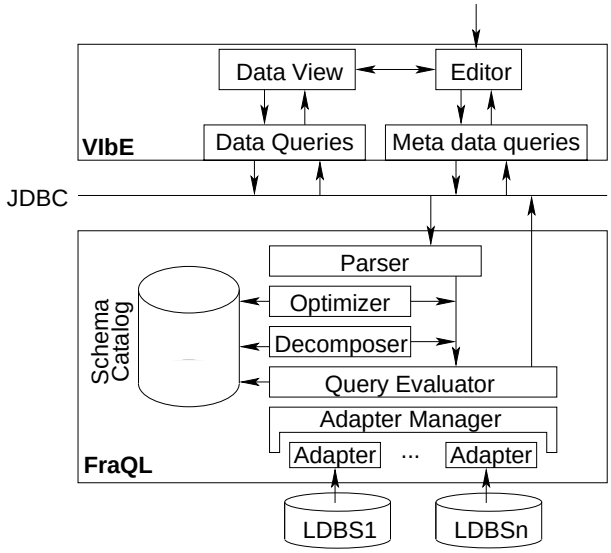

Figure 9: Architecture example-driven integration

The tool combines interactive query generation and evaluation with the conflict resolution. It takes as input the preliminary integrated schema that have to be mapped to the local schemas. Therefore following steps are supported: selection of relevant sources, creating visually the

[5] **V**isual **I**ntegration **b**y **E**xample

isbn	*title*	*category*	*dealer*	*price*	} *relation schema*
		`@map( ... )`		`* 1.18`	} *mapping row*
123-XX	Database Systems	Computer Science	Amazon.com	21.90	} *data view*
234-YY	Data Warehouses	Computer Science	B & N	40.00	
231-ZZ	Web Databases	Computer Science	Books & More	31.00	

Figure 10: Import view editor

integration steps and analyzing result data visually. The visual input paradigm was chosen to support the user in creating and analyzing the mapping.

The system tries to combine conflict resolution and analysis phase as tightly as possible. So the defined views are evaluated as early as possible and the user can check the steps early.

VIBE provides several views on the data from the local sources as well as on intermediate integration results. For example, the *import view editor* displays the imported relation together with mapping information (Fig. 10).

In the heading of the table the global attribute names defined by the specified type of the relation are shown. The second row contains the mapping definition. Here, the name of the corresponding local attribute is given. If a mapping function or a mapping table is required, its name has to be inserted into the appropriate column. In addition, an expression can be entered, that is automatically translated into an user-defined function. In the rows below the mapping row the database integrator can specify QbE-like selection queries. These queries are evaluated and the mapping is applied to the results. In this way, one receives direct feedback of the defined mapping by inspecting the data.

Data conflicts are discovered by using a so-called *conflict map*. This map is constructed as follows: For a union as integration operation an outer join is computed and for each tuple appearing in both input relations (which is determined by comparing the primary keys) the corresponding attribute values are compared. Both values are presented in a single cell of the map, where the color depends on the comparison result. If both values are equal the color of the cell is white, otherwise red. Therefore, a red cell denotes an attribute conflict.

For a join as integration operation the map is constructed by applying an outer join, too. In addition the coloring for the union operation, a further kind of conflict is considered. Null values of attributes appearing in only one input relation are presented as yellow cells. So, these cells can indicate key equivalence conflicts. In fact, not the actual values of the resulting tuples are important, but the colors. Therefore, a compact representation of conflict spots is possible. The user can zoom into the overview map and select points of interests for further examinations.

4.2 SigmaBench

Here we sketch the tool support for the GIM approach. In the future we will try to combine the tool support in correspondence to the combined method.

We implemented the federated database design tool **SIGMA**$_{Bench}$ (see Fig. 11). 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 the file structure analysis if local databases are stored in files. 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 schemas from an Oracle database to our repository. It uses the catalogue tables of the Oracle system and currently con-

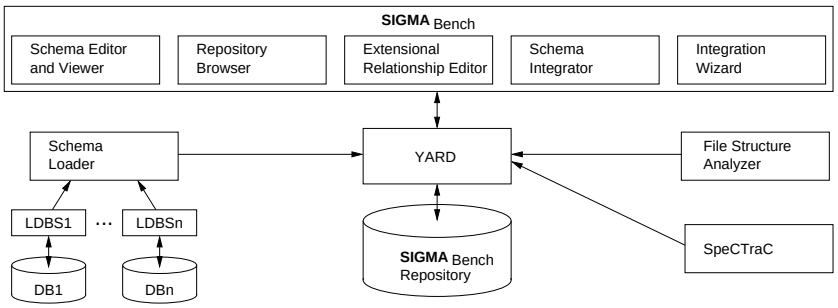

Figure 11: Architecture of the **SIGMA**$_{Bench}$

siders 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 schemas for storing them in the repository. Of course, in "real" integration scenarios the schemas 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 an extension diagram which correlates disjoint extensions and classes of the participating schemata. 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 step for the federated schema generation. It includes the *assertion generator* which currently supports attribute-constant-comparisons and specialization relationships. SPECTRAC[6] 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.

4.3 Towards an Integrated Tool Set

The previously described integration tools VIBE and **SIGMA**$_{Bench}$ provide advanced support for data integration, but follow only their own integration paradigm. **SIGMA**$_{Bench}$ focuses on a complete formal approach, whereas VIBE enables a data-driven and a more pragmatic way. However, we believe that for real-world integration tasks both scenarios with and without a given integrated schema are important and an alternate proceeding seems to be the best solution.

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

For instance, a first coarse global schema – perhaps based on a domain reference model – is given, which has to be extended and refined during the integration of the individual sources. This approach could benefit from a combined usage of both tool sets. From a technical point of view the integration of these tools is entailed with low effort only, because both systems are implemented in Java and use a standard database API. Moreover, a multidatabase language like FRAQL provides all the features required for accessing the sources, defining global views as well as mapping and resolving conflicts.

Following the method described in section 3 we can sketch the integration process using the tools in a combined manner as follows:

1. The global schema is designed using a interactive schema editor. The result of this step is a set of global types and their relationships defined in FRAQL.

2. For each participating local database vertical conflicts between the local schema and the global schema which is represented by the types from step 1 are resolved. For this purpose the VIBE import view editor is used which displays the imported relation together with the mapping information.

3. In the next step, the extensional analysis is performed using the **SIGMA**$_{Bench}$ extensional relationship editor, which has to be extended by extension checks discussed in section 3.3. This step is additional supported by the VIBE data views including the described conflict map. The result of this phase is a set of global views implementing the types resulting of step 1 and accessing the intermediate views of step 2.

If the assumed extensional correspondences in step 3 are not fulfilled by the current data several solutions are possible. First, the designer could return to step 2 refining the mapping. Second, additional sources could be considered if available and finally, restarting with step 1 the global schema could be adjusted in order to meet the requirements.

Particularly, during step 2 and 3 the VIBE tools in combination with the multidatabase query system provide a comprehensive view on the intermediate integration results. Even if these views are only snapshots of the current database state, in many cases the remaining conflicts can be visualized and as a results identified and resolved by the designer.

5 Related Work

The problem of schema integration in the context of multidatabases is a relatively old problem. There is a huge number of publications in this area. Integrations problems and solution approaches in loosely coupled FDBS are discussed, for example, in [Wie92, ZHK96].

Very important for a schema integration is the choice of the right common data model in which the schema integration is performed. The suitability of different data models as common data model is discussed in [BLN86, SCG91, HB96]. The favorite data model is usually an object-oriented data model due to its semantical richness. As common data model for homogenization in the GIM approach we use the Generic Integration Model GIM which enables an efficient algorithm to derive an integrated schema in a user-friendly data model. The data model GIM was firstly introduced in [SS95, SS96a].

The extensional conflict as one main conflict is topic of many publications, e.g. [DH84, Mot87, MNE88, Bra93, SGN93, TS93, KS95, NS96]. They usually resolve this conflict directly in an object-oriented model by using specialization. The original classes are often classes of the integrated schema enriched by new super-/subclasses and specialization relationships

among them. [DS96], for example, suggests many operations to resolve a conflict between two classes. Problems arise, however, if two specialization hierarchies with many classes need to be integrated. The mentioned approaches generate in this case very complex schemata. Furthermore, different variants of conflict resolution are often possible. There are no strict rules which help the designer. Therefore, the process of integrating specialization hierarchies is usually very hard for the designer and produces often a huge number of new classes.

In contrast to these approaches, basing on a correct extensional analysis we firstly decompose class extensions into base extensions and then use mechanisms of concept analysis. These mechanisms perform the composing of base extensions to extensions of global classes. We try to derive minimal schemata by finding maximal rectangles and by further reductions of superfluous classes.

The idea to use mechanisms of the formal concept analysis for the design of object-oriented database is not new. [YLCB96], for example, uses this technique to generate a class hierarchy depending on an intensional analysis. In contrast to our approach, however, [YLCB96] does not consider extensional relationships and is therefore not useful for schema integration. In [SS96a, SS96b, SC97, SS98] we described how to decompose class extensions for schema integration. This decomposition enables us to use mechanisms of formal concept analysis for schema integration. Our improved algorithm was firstly published in [SS98].

Query languages supporting the integration of heterogeneous sources are particularly multidatabase languages like MSQL [GLRS93], SQL/M [KGK+95] and SchemaSQL [LSS96]. MSQL provides basic features for accessing schema labels and converting them into data values. SQL/M addresses mainly description conflicts by providing mechanisms for scaling and unit transformation. More advanced conflict resolution is addressed for example by the restructuring techniques proposed in SchemaSQL supporting the specification of relation with data dependent output schemas. The language FRAQL extends these by additional resolution techniques for attribute and structure conflicts as well as data conflicts.

Examples of systems addressing reconciliation are systems like Pegasus [ASD+91], the IBM DataJoiner [VZ98], Garlic [TAH+96] or TSIMMIS [GPQ+97]. Pegasus uses a functional object-oriented data manipulation language called HOSQL with non-procedural features, DataJoiner is based on DB2 and therefore provides essentially standard SQL features for conflict resolution. Garlic is a database middleware which uses a wrapper architecture to access different sources. Garlic provides an object-oriented extensions to SQL which allow for instance extended conflict resolution. TSIMMIS is a mediator system. The mediator is specified by a set of rules. Each rule maps a set of source objects into a virtual mediator object. In this way, conflicts are resolved by defining appropriate rules.

Data-driven integration tools are for instance Clio [MHH+01] and Potter's Wheel [RH01]. Clio is a semi-automatic tool for evaluating schema mappings by using example data. After user analysis the schema mapping can be refined. Potter's Wheel uses scalable spreadsheets to do data cleansing and transformation. The actual data is used for interpreting the transform results.

6 Conclusions

Supporting different integration scenarios with a single method is often involved with limitations. So, in this paper we sketched the combination of a formal method for schema integration with a more pragmatic example-driven method. We believe that this approach is particular useful in scenarios where a global schema is given and mappings can be checked using available

data. We further discussed how our available tools can be used in an integrated, iterative manner. For further work, we plan to extend the approach by additional data-driven analysis tools, e.g. for detecting wrong assignments of objects to extensions during the extensional analysis. A second important task will be the integration of our tools into a homogeneous tool set guiding the designer through the integration process.

References

[ASD+91] R. Ahmed, P. D. Smedt, W. Du, W. Kent, M. A. Ketabchie, W. A. Litwin, A. Rafii, and M.-C. Shan. The Pegasus Heterogeneous Multidatabase Systems. *IEEE Computer*, 24(12):19–26, December 1991.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[Bra93] S. E. Bratsberg. *Evolution and Integration of Classes in Object-Oriented Databases*. Dissertation, The Norwegian Institute of Technology, University of Trondheim, June 1993.

[DH84] U. Dayal and H. Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Transactions on Software Engineering*, 10(6):628–644, November 1984.

[DS96] Y. Dupont and S. Spaccapietra. Schema Integration Engineering in Cooperative Databases Systems. In K. Yetongnon and S. Hariri, editors, *Proc. of the 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems, PDCS'96, Dijon, France, September 1996*, pages 759–765, Six Forks Road, Releigh, NC, 1996. International Society for Computers and Their Application.

[GLRS93] J. Grant, W. Litwin, N. Roussopoulos, and T. Sellis. Query Languages for Relational Multidatabases. *The VLDB Journal*, 2(2):153–171, April 1993.

[GPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, March/April 1997.

[GW98] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer-Verlag, Berlin/Heidelberg, 1998.

[HB96] A. R. Hurson and M. W. Bright. Object-Oriented Multidatabase Systems. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications*, chapter 1, pages 1–36. Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[KGK+95] W. Kelley, S. Gala, W. Kim, T. Reyes, and B. Graham. Schema Architecture of the UniSQL/M Multidatabase System. In W. Kim, editor, *Modern Database Systems*, chapter 30, pages 621–648. ACM Press, New York, NJ, 1995.

[KS95] W. Klas and M. Schrefl. *Meta Classes and Their Applications — Data Model Tailoring and Database Integration*, volume 943 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1995.

[LNE89] J. A. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, April 1989.

[LSS96] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. SchemaSQL - A Language for Interoperability in Relational Multi-database Systems. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of the 22nd Int. Conf. on Very Large Data Bases, VLDB'96, Bombay, India, September 3–6, 1996*, pages 239–250, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

[MHH+01] Rene J. Miller, Mauricio A. Hernndez, Laura M. Haas, Ling-Ling Yan, C. T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 30(1):78 – 83, March 2001.

[MNE88] M. V. Mannino, B. N. Navathe, and W. Effelsberg. A Rule-based Approach for Merging Generalization Hierarchies. *Information Systems*, 13(3):257–272, 1988.

[Mot87] A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13(7):785–798, July 1987.

[NS96] S. Navathe and A. Savasere. A Schema Integration Facility Using Object-Oriented Data Model. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications*, chapter 4, pages 105–128. Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[RH01] Vijayshankar Raman and Joseph M. Hellerstein. Potters Wheel: An Interactive Framework for Data Cleaning and Transformation. Working draft, 2001.

[SC97] I. Schmitt and S. Conrad. Restructuring Class Hierarchies for Schema Integration. In R. Topor and K. Tanaka, editors, *Database Systems for Advanced Applications '97, Proc. of the 5th Int. Conf., DASFAA'97, Melbourne, Australia, April 1–4, 1997*, pages 411–420. World Scientific Publishing, Singapore, 1997.

[SCG91] F. Saltor, M. Castellanos, and M. Garcia-Solaco. Suitability of Data Models as Canonical Models for Federated Databases. *ACM SIGMOD Record*, 20(4):44–48, December 1991.

[Sch98] I. Schmitt. *Schema Integration for the Design of Federated Databases*, volume 43 of *Dissertationen zu Datenbanken und Informationssystemen*. infix-Verlag, Sankt Augustin, 1998. (In German).

[SCS00] K.-U. Sattler, S. Conrad, and G. Saake. Adding Conflict Resolution Features to a Query Language for Database Federations. *Australian Journal of Information Systems*, 8(1):116–125, 2000.

[SGN93] A. P. Sheth, S. K. Gala, and S. B. Navathe. On Automatic Reasoning for Schema Integration. *Int. Journal of Intelligent and Cooperative Information Systems*, 2(1):23–50, 1993.

[SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, July 1992.

[SS95] I. Schmitt and G. Saake. Managing Object Identity in Federated Database Systems. In M. Papazoglou, editor, *OOER'95: Object-Oriented and Entity-Relationship Modeling, Proc. of the 14th Int. Conf., Gold Coast, Australia, December 1995*, volume 1021 of *Lecture Notes in Computer Science*, pages 400–411, Berlin, 1995. Springer-Verlag.

[SS96a] I. Schmitt and G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In B. Thalheim, editor, *Conceptual Modelling — ER'96, Proc. of the 15th Int. Conf., Cottbus, Germany, October 1996*, volume 1157 of *Lecture Notes in Computer Science*, pages 195–210, Berlin, 1996. Springer-Verlag.

[SS96b] I. Schmitt and G. Saake. Schema Integration and View Generation by Resolving Intensional and Extensional Overlappings. In K. Yetongnon and S. Hariri, editors, *Proc. of the 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems (PDCS'96), Dijon, France, September 1996*, pages 751–758, Six Forks Road, Releigh, NC, 1996. International Society for Computers and Their Application.

[SS98] I. Schmitt and G. Saake. Merging Inheritance Hierarchies for Database Integration. In M. Halper, editor, *Proc. of the 3rd IFCIS Int. Conf. on Cooperative Information Systems, CoopIS'98, August 20–22, 1998, New York, USA*, pages 322–331, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[SS99] I. Schmitt and G. Saake. Integrating Database Schemata using the GIM Method. Preprint 20, Fakultät für Informatik, Universität Magdeburg, 1999.

[ST98] I. Schmitt and C. Türker. Refining Extensional Relationships and Existence Requirements for Incremental Schema Integration. In G. Gardarin, J. French, N. Pissinou, K. Makki, and L. Bougamin, editors, *Proc. of the 7th ACM CIKM Int. Conf. on Information and Knowledge Management, November 3–7, 1998, Bethesda, Maryland, USA*, pages 322–330, New York, 1998. ACM Press.

[TAH+96] M. Tork Roth, M. Arya, L. M. Haas, M. J. Carey, W. Cody, R. Fagin, P. M. Schwarz, J. Thomas, and E. L. Wimmers. The Garlic Project. In H. V. Jagadish and I. S. Mumick, editors, *Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada*, volume 25 of *ACM SIGMOD Record*. ACM Press, June 1996.

[TS93] C. Thieme and A. Siebes. Schema Integration in Object-Oriented Databases. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Advanced Information System Engineering, Proc. of the 5th Conf., CAiSE'93, Paris, France, June 1993*, volume 685 of *Lecture Notes in Computer Science*, pages 55–70, Berlin, 1993. Springer-Verlag.

[VZ98] Shivakumar Venkataraman and Tian Zhang. Heterogeneous Database Query Optimization in DB2 Universal DataJoiner. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 685–689. Morgan Kaufmann, 1998.

[Wie92] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 25(3):38–49, March 1992.

[YLCB96] A. Yahia, L. Lakhal, R. Cicchetti, and J. P. Bordat. iO_2: An Algorithmic Method for Building Inheritance Graphs in Object Database Design. In B. Thalheim, editor, *Conceptual Modelling — ER'96, Proc. of the 15th Int. Conf., Cottbus, Germany, October 1996*, volume 1157 of *Lecture Notes in Computer Science*, pages 422–437, Berlin, 1996. Springer-Verlag.

[ZHK96] G. Zhou, R. Hull, and R. King. Generating Data Integration Mediators that Use Materialization. *Journal of Intelligent Information Systems*, 6(2/3):111–133, June 1996.

[Zlo77] M.M. Zloof. Query-by-Example: A Data Base Language. *IBM Systems Journal*, 16(4):324 – 343, 1977.