Schema Integration and View Derivation by Resolving Intensional and Extensional Overlappings

Ingo Schmitt & Gunter Saake
Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Magdeburg, PF 4120, 39016 Magdeburg, Germany

Abstract

The integration of schemata is a very essential but also a very complex task in a federated database environment. Especially the integration of different inheritance hierarchies into one hierarchy is not satisfactorily solved up to now. We show, why some existing approaches fail and propose a new algorithm, which further meets the demand for complete, correct and minimal integrated schemata. Furthermore, with respect to the well-known concept of logical data independence, our algorithm supports the derivation of different view-dependent external schemata. External object-oriented schemata can be automatically generated by our algorithm.

Our approach can be applied to scenarios different from the integration problem, e.g. for schema evolution, views in object-oriented database systems, and classification problems.
Keywords: database federation, schema integration, view derivation, inheritance hierarchy, extensional and intensional conflicts

1 Introduction

In many large organizations, different legacy data management systems are maintained and operated. These data management systems and the data managed by them have developed independently from each other. The data management systems are often database management systems or merely file-based systems differing in several aspects such as data model, query language, system architecture, etc. as well as the structure and semantics of the data managed. In this context the term 'heterogeneity of data management systems' is commonly used. As a rule, applications for specific needs continue to be based on such systems. More recent applications often require access to distributed databases but their implementation fails due to heterogeneity. *Federated database management systems* (FDBMS, cf. [20]) are designed to solve this problem.

An FDBS offers a homogeneous integrated schema. There exist transformations from the integrated schema to the local schemata. In the first step of the integration process the data model heterogeneity can be resolved by transforming the local schemata into the canonical data model of the FDBS. In the second step schematic heterogeneity is solved. In general, this step is a very complex task due to many kinds of conflicts. Therefore, different integration methodologies were proposed in [2, 12, 21, 7, 15, 14].

There are different requirements attributed to the integration of local schemata into an integrated schema (adopted from [2] page 337):

- *Completeness and Correctness*: The integrated schema must contain all concepts present in any component schema correctly. The integrated schema must be a representation of the union of the application domains associated with the schemas.
- *Minimality*: If the same concept is represented in more than one component schema, it must be represented only once in the integrated schema.
- *Understandability*: The integrated schema should be easy to understand for the designer and the end user. This implies that among the several possible representations of results of integration by a data model, the one that is (qualitatively) the most understandable should be chosen.

Furthermore, an integration methodology has to support *logical data independence*. That is, it must be possible to derive different views (external schemata) from an integrated schema. The generation of a view has to fit to the understanding of a specific user group. Users want to influence the generation of their external schema.

These requirements and especially the support of different federated views was not sufficiently met by the proposed methodologies.

The approach we present in this paper differs from most existing proposals in using a *semantically poor data model* as canonical data model. The data model used for the integration is based on a few orthogonal data structuring concepts, namely classes, attributes and object identification. Furthermore, it has fundamental integrity constraints capturing part of the semantics lost during the transformation towards a semantically poor data model.

We show that this approach allows the use of basic information on intensional and extensional overlappings of original classes for an automatic integration and application specific view (or external schema) generation. Especially, the integration of *inheritance hierarchies* in object-oriented approaches is supported by a view derivation algorithm controlled by marking classes. Classes which are marked as preferred will strongly influence the structure of the derived inheritance trees, that is, they will be located close to the roots of the derived inheritance trees.

2 Motivation: Deficiencies of Integration Approaches

Most existing approaches for integrating class hierarchies fail in exploiting information concerning extensional overlappings combined with intensional overlappings. Instead, they only examine either the intensional overlappings, i.e., they only consider attribute declarations, or extensional overlappings as input for detecting similarities between only two classes for integration decisions.

Figure 1 shows two very simple class hierarchies of object-oriented databases. Aim of the integration process is to find a minimal but complete schema for an integrated object-oriented database.

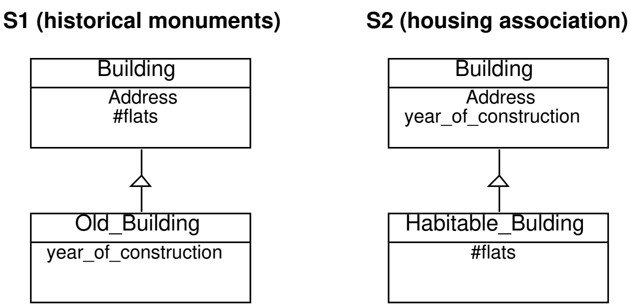

Figure 1: Example Class Hierarchies for Integration

One part of the integration process is the analysis of possible *extensional overlappings* between classes from different schemata. This analysis results in constraints between class populations (for example: disjointness, intersection, inclusion, identical population). For our example, we detect extensional overlappings as shown in Figure 2. For instance, all 'new buildings' (`S1.Buildings` not specialized to old buildings) are included in the second database as habitable buildings.

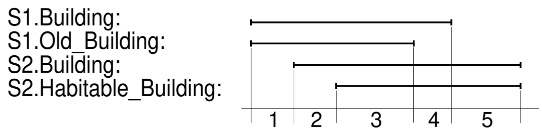

Figure 2: Extensional Overlapping of Input Classes

Most existing integration approaches [13, 12, 21, 3, 7, 22, 15, 8, 14] do only mark pairs of classes with a possibly non-empty population intersection as candidates for integration. Additionally, attributes may be marked to be corresponding.

The resulting integration *assertions* can be listed in the notation proposed by [22]:

S1•Building ∩ S2•Building
Attribute correspondence:
S1•Building•Address ≡ S2•Building•Address

S1•Building ∩ S2•Habitable_Building
Attribute correspondence:
S1•Building•#flats ≡ S2•Habitable_Building•#flats

S1•Old_Building ∩ S2•Building
Attribute correspondence:
S1•Old_Building•year_of_construction ≡
S2•Building•year_of_construction

S1•Old_Building ∩ S2•Habitable_Building

Most existing integration approaches resolve extensional overlappings between two classes by upward inheritance. For example, applying upward inheritance to the first assertion results in a class hierarchy as shown in Figure 3. Both original schemata result in one branch of the new class hierarchy; both branches are connected by a new base class as root of the hierarchy.

However, by carefully analyzing Figure 2 it becomes obvious that the instances of all classes have all three attributes. For example, new buildings are always habitable and have therefore the attribute `year_of_construction`. This information cannot be derived from the listed assertions.

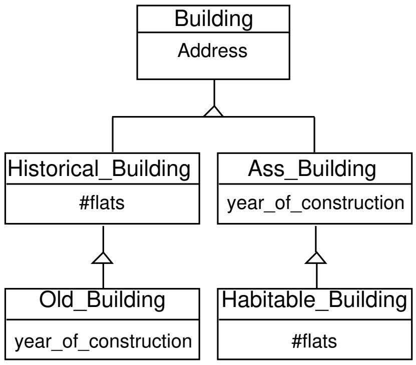

Figure 3: Integrated Schema by using Upward Inheritance

Exploiting these information, we can combine all instances into one class having all three attributes. However, classification information is lost and has to be added using a classification attribute `discriminant` (cf. [8]) which specifies the overlapping area (i.e. 1,2,3,4, or 5 as depicted in Figure 2). Using this attribute, the original class hierarchies can be derived as views. The resulting integrated schema is shown in Figure 4.

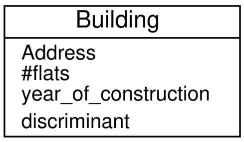

Figure 4: Well Integrated Schema

Our small example shows that it makes sense to consider more information on extensional overlappings than it is done in classical approaches. Using these additional assertions, the resulting integrated schema may even become simpler than the original database schemata.

3 Generic Integration Model

The right choice of the canonical data model is essential to the design process of an FDBS. This decision influences the quality of the integration step, which is the most difficult step of the federated database design process. Suggested common data models vary from semantically poor data models, e.g. the relational data model, to semantically rich data models, e.g. an object-oriented data model. Most of the recent proposals prefer an object-oriented data model. It is often argued that only a semantically rich data model can be the right canonical data model because no or only little semantics is lost during the transformation from an object-oriented local schema to the canonical data model (see also [16]). On the other hand, a semantically rich data model causes more heterogeneity on the schema level than a semantically poor data model does (cf. [2]). This heterogeneity on schema level in turn increases the complexity of the integration process (cf. object-oriented integration methodologies described in [21, 15, 14, 8] and integration in the entity-relationship model described in [7]). Therefore, some researchers ([1, 4]) do not try to find a global integrated schema. Instead of this, they propose a loosely coupled approach.

We chose another way out. In order to reduce the complexity, we propose the usage of a semantically poor data model. For best support of the integration process we designed an own data model called the Generic Integration Model (GIM is introduced in [17][1]). In GIM, only schema information which is necessary for the integration process can be expressed by means of GIM concepts whereas schema information which defines a specific view of an application cannot be expressed. Thus, a schema expressed in GIM offers the flexibility for deriving different external schemata.

Since the presentation of GIM is not the focus of this paper we only sketch the concepts of GIM. GIM supports the idea of object identity (see also [9, 18]). The main difference to other data models (which support inheritance) is the demand for non-overlapping class extension. The extensions of any two classes have to be either disjoint or identical. Subset relations or partial overlappings between class extensions are not allowed.

Due to this restriction, GIM cannot function as interface to applications. In consequence, external schemata expressed in another data model than GIM have to be derived from the integrated schema.

Too many schema transformations in an FDBS decrease its performance at run-time. We propose to use a design tool, which is separated from the FDBS. This tool can help to develop the schema hierarchy graphically. Transformation information for specific FDBSs can be derived from the designed schema hierarchy after transformation steps have been optimized or unnecessary transformation steps have been removed.

We present GIM schemata in form of a diagram (cf. Figure 5). In order to understand GIM diagrams we have to explain the semantics of its graphical elements. The horizontal dimension refers to the extensional as-

[1]A complete formalization of GIM will be given in a forthcoming paper.

pect whereas the vertical dimension refers to the intensional aspect of a class. The indices of the rows refer to enumerated groups of attributes which are mutually disjoint whereas the indices of the columns refers to enumerated extensional partitions resulting from decompositions of extensional overlappings. A cross inside the diagram means, that each object of this extensional partition (column) has valid values for the attributes of this intensional group (row). A possible class itself is represented by a rectangle.

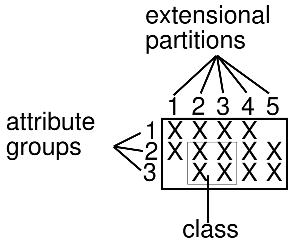

Figure 5: GIM Diagram Conventions

4 Schema Decomposition

The original database schemata to be federated have to be transformed into GIM to enable their integration. This transformation is a complex task and can be divided into several tasks [6]. For simplicity, we combine here the process of transformation to GIM and the process of integration into a GIM schema into one process. In principle, the input database schemata have heterogeneous data models, and naming, attribute and constraint conflicts have to be detected and solved. A taxonomy of conflicts is given in [19, 10].

For our discussion, we use a very simplified example focusing on extensional and intensional overlappings. We assume that other conflicts are resolved (e.g. meta conflicts [11, 5] and attribute conflicts [12]). Both schemata are object-oriented, and attributes with identical name have identical semantics. The example is taken from two library applications and is shown in Figure 6.

Using the library example, we will now discuss the integration process informally. A more detailed discussion of the following steps is given in [6]. As a first step, we analyze the extensional overlappings of the three classes. This can be done for example using data inspection and key constraint analysis ([23]) and must be assisted by an expert of the application domain. The result of the *extension analysis* is shown in Figure 7.

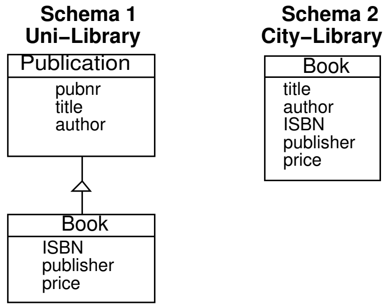

Figure 6: Two heterogeneous Schemata

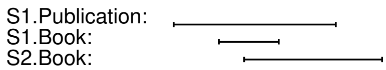

Figure 7: Extensional Overlapping

S1.Publication and S2.Book can overlap extensionally whereas the extension of class S1.Book is always a subset of the extension of class S1.Publication.

Based on the extension analysis, we then perform an *extension decomposition*. In this step, we generate a partition of all class instances based on all possible non-empty intersections of the input class populations. After performing this partition, we have five disjoint classes as shown in Figure 8. For each of these partition classes we have a unique mapping to a combination of input classes and vice versa.

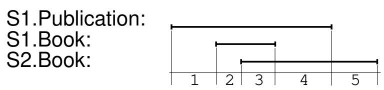

Figure 8: Extensional Decomposition

Besides the extensional part the *intensional* dimension has to be analyzed, too. As a result of the *intension analysis* we detect the intensional overlappings as shown in Figure 9.

The *intensional decomposition* results in the following attribute partition:

```
1: pubnr
2: title, author
3: publisher, ISBN, price
```

The extensional and intensional decomposition can be represented in a diagrammatic form as shown in Figure 10. The horizontal dimension enumerates the five partitions resulting from the extensional decomposition. The vertical dimension is labeled by the three attribute partitions.

This notation can be directly used to derive view classes following the GIM notation. Each rectangle

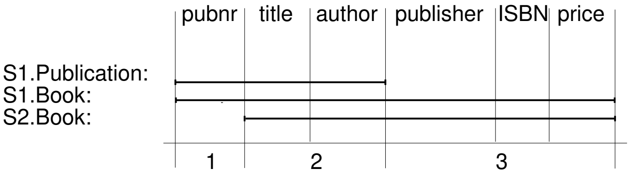

Figure 9: Intensional Overlapping

	1	2	3	4	5
1	X	X	X	X	
2	X	X	X	X	X
3		X	X	X	X

Figure 10: Decomposed Schema

completely filled with crosses is a candidate for a view class. For example, the middle row is a candidate for a class containing all instances and having the attributes `title` and `author`.

Candidates for subclasses are rectangles which are horizontally included in another class rectangle but vertically completely cover it. This is graphically depicted in Figure 11. With other words, subclasses have a subset of the instance population and a superset of the attributes.

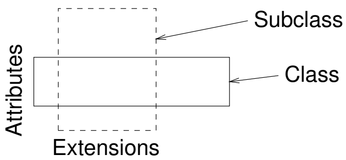

Figure 11: Class and Subclass Candidates in GIM

However, the order of both attribute and class labels in the diagrammatic notation is arbitrarily chosen. As a result, a different column and row order may show different rectangles as candidates corresponding to different class hierarchies for view classes.

5 Algorithm for Deriving External Schemata

The derivation of external schemata from the partitioned diagrammatic representation is based on detecting rectangles as class and subclass candidates for an external schema. As already mentioned, the order in both dimension is irrelevant and has no semantics. In principle, we would have to construct representations for all permutations of both dimensions and identify rectangles as class candidates. However, this is computationally too expensive.

For small examples the federation designer can directly identify class candidates because the designer knows about the application semantics. For large schemata, an automatic process is necessary. It can be easily seen that an external schema which has to be equivalent to a local schema can be derived automatically from the partitioned diagram by compositions, which undo former decompositions. Such an equivalent external schema facilitates the migration of local applications to the external schema.

We will present an algorithm for automatic derivation of external schemata which can be influenced by the designer. The steps of the algorithm are explained using our library example.

As first step the designer has to order the extensional dimension by *priority*. The order of the extensional dimension influences the result of the derivation process: classes corresponding to extensions with higher priority (leftmost side) will be located closer to the root of the inheritance tree of the resulting external schema. The first diagram of Figure 12 gives the highest priority to extension `1` corresponding to publications in `S1` which are no books and not included in the instance set of `S2`. Generally, the extensions of schema `S1` are given higher priority than those of `S2`.

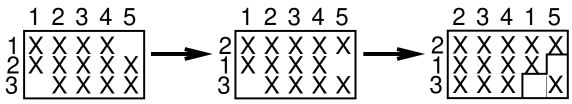

Figure 12: Publication-oriented Diagram

To derive rectangles as class candidates, we *sort* the *rows* of the diagram. Each row is interpreted as a binary number. These numbers are sorted in descending order. The result of the sort operation is shown in the middle diagram of Figure 12.

As next step, we analogously sort the *columns* of the diagram where the crosses of the higher rows correspond to higher bits of the binary representation. We have to repeat alternatively both sort operations until columns and row are sorted simultaneously. The result of this step is depicted in the right diagram of Figure 12. As we can see, crosses are clustered in the left upper corner of the diagram building the shape of a triangle. This triangle can be interpreted as composed of rectangles building a branch of a class hierarchy as depicted in Figure 13.

This branch constitutes a part of the inheritance tree for the external schema. For our example, this branch contains three classes. The first class is the root of the inheritance tree. Extensions and intensions of these classes can be derived from the information

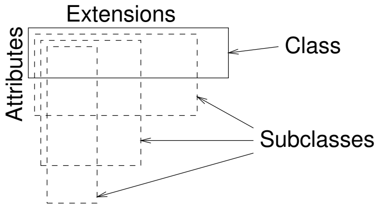

Figure 13: Subclass Candidates in Triangle Shape

shown in Figures 9 and 8. Using these information, a designer has to find class names corresponding to the semantics of these derived classes. As a result we have the following three classes:

```
Extension  Intension  Class
--------------------------------
2,3,4,1,5  2          Publication
2,3,4,1    2,1        Uni_Pub
2,3,4      2,1,3      ISBN_Pub
```

After these steps, we may have some crosses left. In our example, we have one cross at position (5,3). In general, we may have more than one cross left. For the remaining crosses we have to construct the remaining branches of the inheritance tree by recursively applying our algorithm.

The area below the triangle has to be partitioned by vertical lines starting from the step corners of the triangle. In Figure 12 the partition is generated by one thin vertical line in the right diagram. The first area encompasses position (1,3) and the second area encompasses positions (5,1) and (5,3). Each resulting partitioned area can be considered as an isolated subdiagram and if it contains more than one cross, it has to be transformed into the triangle form by a further intensional and extensional sorting. These triangles correspond to other branches of the constructed inheritance tree. They are located at specific positions in the diagram: the most upper crosses are located vertically below a step of the triangle shape. This triangle step determines the position where the new branch has to be added into the inheritance tree. This procedure has to be applied recursively to each subarea or subsubarea until all crosses are represented in external classes.

In our example, we have only one cross corresponding to a class located directly below the root of the inheritance tree:

```
Extension Intension  Class
------------------------------
5         2,3        City_Book
```

The new class `City_Book` becomes subclass of `Publication`. As a last step, we have to analyze whether all columns are distinguished by their mapping to subclasses. If not, we have to add an artificial *discriminant* attribute. In the generated external schema we are not able to distinguish objects from extensional partitions 2, 3, or 4. Hence, we have to enhance the class `ISBN_Pub` by the *discriminant* attribute.

Figure 14 shows the inheritance tree computed by the algorithm. The designer has guided the algorithm by giving priorities to extensions only. However, it is the task of the designer to find appropriate names for the constructed classes.

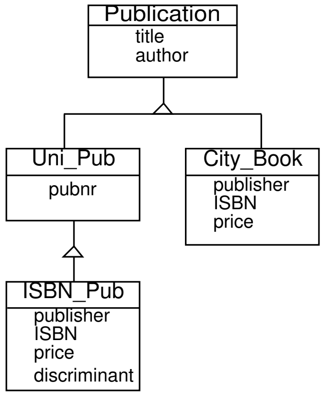

Figure 14: Publication-oriented Schema

The result of the algorithm is a collection (in our small example only one) of inheritance trees having simple inheritance only. Each instance is mapped to exactly one class. The extensions of all direct subclasses of a class are mutually disjoint.

The priorities of the input extension order influence the results of the algorithm. If the extensional partitions of a local class have the highest priority then there is exactly one external class in the resulting inheritance tree which corresponds to the local class.

As an example, we order the extensions in such a way that all extensions which correspond to 'books' have the highest priority. The result of the sort steps is shown in Figure 15

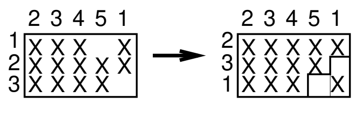

Figure 15: Book-oriented Diagram

The derived inheritance tree is depicted in Figure 16. In this case it is different from the former tree derived. In contrast to the first tree, all books are clustered in one branch of the inheritance tree.

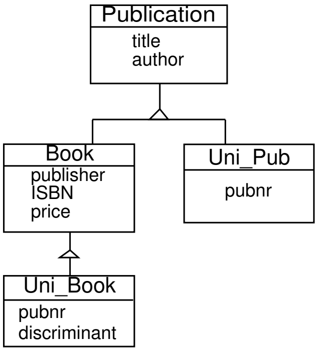

Figure 16: Book-oriented Schema

6 Conclusions

The presented framework offers a practicable algorithm to derive class hierarchies, which meet the demand for minimality, correctness, completeness, and understandability. The integrated schema in GIM is correct and complete but neither easily understandable nor minimal. However, it is suited to derive external schemata on it following our algorithm. A generated external schema is complete because it covers all crosses of the GIM schema. Our algorithm generates correct (following the ideas presented in Figure 13) inheritance trees. Minimality is achieved by the sorting operations. These operations move most crosses to the left uppermost corner. Hence, we are able to generate few external classes covering many "crosses" of the GIM schema. The view composition algorithm is computationally simple and allows to influence the resulting inheritance hierarchy by giving priorities to input extensions.

The presented methods can be applied to scenarios different from the integration problem, e.g., for schema evolution, deriving views in object-oriented database systems, and classification problems.

Future work will concentrate on integrating more semantical assertions (e.g. integrity constraints) into the framework. Moreover, the aspect of multiple inheritance in derived inheritance trees has to be investigated. Our algorithm enable the designer to generate different views. Further work is necessary to assist the designer in generating its "best" view.

The presented work is part of a larger project on federating heterogeneous databases [6]. In this context the integration of the presented methods into a larger design framework will be investigated. Especially, the role of constraints in federated database design is one focal point of the project.

A design tool supporting the presented methods and algorithms is currently under development.

Acknowledgements

The authors are grateful to Michael Höding, Can Türker and Stefan Conrad for helpful comments.

References

[1] T. Barsalou, A. Keller, Siambela N., and G. Wiederhold. Updating Relational Databases through Object-Based Views. In J. Clifford and R. King, editors, *Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data, Denver, Colorado, SIGMOD RECORD 20(2)*, pages 248–257. ACM Press, June 1991.

[2] 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.

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

[4] S. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heteregenous Information Sources. In *Proc. of IPSI Conf.*, 1994.

[5] J. Chomicki and W. Litwin. Declarative Definition of Object-Oriented Multidatabase Mappings. In T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 375–392. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[6] S. Conrad, M. Höding, S. Janssen, G. Saake, I. Schmitt, and C. Türker. Integrity Constraints in Federated Database Design. Preprint Nr. 2, Fakultät für Informatik, Universität Magdeburg, April 1996.

[7] Y. Dupont. Resolving Fragmentation Conflicts in Schema Integration. In P. Loucopoulos, editor,

Entity-Relationship Approach — ER'94, Proc. of the 13th Int. Conf. on the Entity-Relationship Approach, Manchester, UK, pages 513–532. LNCS 881, Springer-Verlag, Berlin, December 1994.

[8] M. Garcia-Solaco, M. Castellanos, and F. Saltor. A Semantic-Discriminated Approach to Integration in Federated Databases. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95), Vienna, Austria*, pages 19–31, May 1995.

[9] S. N. Khoshafian and G. P. Copeland. Object Identity. In N. Meyrowitz, editor, *Proc. of the 1st Int. Conf. on Object Oriented Programming Systems, Languages and Applications (OOPSLA'86), Portland, Oregon, SIGPLAN Notices 21(11)*, pages 406–416. ACM Press, November 1986.

[10] W. Kim, editor. *Modern Database Systems.* ACM Press, New York, 1995.

[11] R. Krishmamurthy, W. Litwin, W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan*, pages 144–151. IEEE Computer Society Press, April 1991.

[12] 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.

[13] S.B. Navathe, R. Elmasri, and J.A. Larson. Integration User Views in Database Design. In *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE'86)*, pages 50–62. IEEE Computer Science Press, January 1986.

[14] E. Pitoura, O. Bukhres, and A. K. Elmagarmid. Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, 27(2):141–195, June 1995.

[15] M. P. Reddy, B. E. Prasad, P. G. Reddy, and A. Gupta. A Methodology for Integration of Heterogeneous Databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):920–933, December 1994.

[16] 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.

[17] I. Schmitt. Flexible Integration and Derivation of Heterogeneous Schemata in Federated Database Systems. Preprint Nr. 10, Fakultät für Informatik, Universität Magdeburg, November 1995.

[18] I. Schmitt and G. Saake. Managing Object Identity in Federated Database Systems. In M. Papazoglou, editor, *Proc. of the 14th Int. Conf. on Object-Oriented and Entity-Relationship Modeling (OOER'95), Gold Coast, Australia*, pages 400–411. LNCS 1021, Springer-Verlag, December 1995.

[19] A. Sheth and V. Kashyap. So Far (Schematically) yet So Near (Semantically). In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems (DS-5), Proc. of the IFIP WG 2.6 Database Semantics Conf., Lorne, Victoria, Australia, November, 1992*, pages 283–312. North-Holland, 1993.

[20] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

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

[22] S. Spaccapietra and P. Parent. View Integration:A Step Forward in Solving Structural Conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, April 1994.

[23] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95), Vienna, Austria*, pages 4–18, May 1995.