Refining Extensional Relationships and Existence Requirements for Incremental Schema Integration[1]

Ingo Schmitt and Can Türker

Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, 39016 Magdeburg, Germany
E-mail: {schmitt|tuerker}@iti.cs.uni-magdeburg.de
Phone: ++49/391/67-12994 Fax: ++49/391/67-12020

February 1998

[1]This work was partly supported by the German Federal State Sachsen-Anhalt under grant number FKZ 1987A/0025 ("Federating heterogeneous data base systems and local data management components for global integrity maintenance").

Abstract

Schema integration as part of the database design as well as of the database integration process has to deal with extensional relationships among classes of the different schemata to be integrated. As a rule, the extensional relationships must be specified by the database designer. Partially, some relationships may also be derived from other relationships. Most existing schema integration approaches consider relationships only between two classes. A correct and complete schema integration, however, can only be performed if all extensional relationships among all classes to be integrated are completely and correctly specified. In this paper we present an incrementally way to support this. Usually, an ad hoc specification of extensional relationships by the database designer is often not feasible. Therefore, we develop a method to derive extensional relationships from an incomplete specification. From these relationships an integrated schema is derived which is only partially correct and can be used for further extensional relationship refinements and corrections. In this way we support an incremental process of schema integration.

Keywords: incremental schema integration, extensional relationships, federated database design, data warehousing.

Acknowledgments: We thanks Michael Höding and Andreas Christiansen for useful comments.

Contents

Chapter 1

Introduction

The process of schema integration is an essential activity for designing non-redundant, semantically correct databases. In classical database design, where schema integration occurs as view integration [NG82, DH84, BC86], a global conceptual schema is derived from the view definitions of the intended applications. On the other hand, in database integration in a distributed or federated database environment [SL90], schema integration aims at deriving an integrated schema which is a virtual view on all database classes to be integrated [Mot87, LNE89, SP94]. This integrated schema provides users a uniform and transparent interface to distributed and possibly heterogeneous databases. In modern database applications such as data warehousing [Wid95] schema integration is also a central issue.

Since the early eighties, schema integration is discussed in both contexts and a lot of approaches have been proposed (for a survey see e.g. [BLN86, PBE95]). A crucial point of schema integration is to overcome the semantic heterogeneity [SK93, GSC96] of the schemata to be integrated. Semantic heterogeneity occurs when one real-world concept is modeled by database designers in different ways. One form of semantic heterogeneity are differences in the extensions of the modeled object classes. The detection and handling of such extensional differences is fundamental for a correct and complete schema integration. *Extensional relationships* among object classes of different schemata (coming from different databases) have to be specified by the database designer who is (supposed to be) an expert of the application domain and knows about the semantics of the schemata to be integrated.

Most existing integration methods, e.g. [BL84, NEL86, MNE88, SPD92, RPRG94, GCS95], consider only *binary* extensional relationships. The approach presented in [DS96] furthermore captures two *n-ary* relationships, the covering and the partitioning correspondence. Nevertheless, this approach does not cover the whole range of possible extensional relationships, too. In general, independently whether an approach supports the specification of arbitrary relationships or not, getting a grasp of all extensional relationships occurring in the universe of discourse is a very hard task for the database designer. Nonetheless, the precise specification of the extensional relationships is a pre-

requisite for a semantically correct integration and thus an indicator for the quality of the integrated schema.

In this paper, we present an approach to schema integration which considers binary as well as n-ary extensional relationships. Further we allow the specification of so-called *existence requirements*. The concepts of existence requirements is introduced in order to exactly state which *base extensions* [SS96a, SS96b] are non-empty and, thus, must be considered by the integration algorithm.

Often, the practical use of schema integration fails because the extensional relationships among all classes are not known or cannot be determined in an ad hoc manner [NS96]. Usually, a "semi-correct and fast" solution is preferred rather than a complete and correct one which requires a huge expense to exactly determine the relationships. Therefore, our approach supports an *incremental* way of integrating schemata by refining the sets of extensional relationships and existence requirements. However, in contrast to other integration approaches, our incremental approach leaves open the chance to derive a semantically correct and complete integrated schema (in case the input information is refined such that is correct and complete).

The paper is organized as follows: Section 2 provides the motivation for this paper by pointing out the importance of considering precise extensional relationships during the schema integration and showing the deficiencies of current proposals. After defining some basic notions and concepts in Section 3, we present our approach in Section 4. There, we further demonstrate in which way changes in the sets of extensional relationships and existence requirements effect the integrated schema. Finally, a discussion and an outlook on future work concludes the paper.

Chapter 2

Motivation

An important step of the pre-integration phase [BLN86] is the analysis of the extensional overlappings among the classes of the different schemata to be integrated. The database designer has to specify — as precise as possible — the extensional relationships among the related classes. The precise definition of extensional relationships is also a prerequisite for dealing correctly with redundancy by defining same-functions between semantically related objects of different classes [SS95].

In most existing integration approaches [BL84, NEL86, SPD92, RPRG94, GCS95] the designer only defines binary assertions for each pair of classes with a possibly non-empty extension intersection. Based on this information these integration methods use *upwards inheritance* (or *generalization*) to integrate overlapping classes. However, these approaches may sometimes result in an integrated schema which deviates too much from the modeled real-world. Often, this is due to fact that these methods do not exactly capture the underlying extensional relationships among the classes to be integrated.

In the following, we introduce an example scenario which we will use throughout the paper to illustrate our method as well as to show the deficiencies of existing integration approaches. As depicted in Figure 2.1, we assume that there are three schemata which have to be integrated.

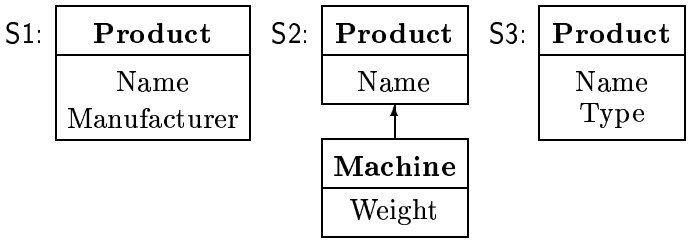

Figure 2.1: Schemata of the Running Example

The extensional relationships among the classes of these schemata are represented in Figure 2.2. In the following, we will use the abbreviations introduced in Figure 2.2 to denote the corresponding classes of the different schemata.

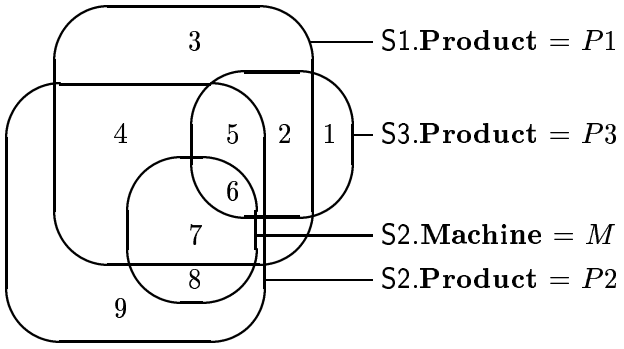

Figure 2.2: Extensional Relationships

After having a closer look at Figure 2.2, we see that the set of all possible objects (universe of discourse) is decomposed by the given extensional relationships into nine base extensions (see also Figure 2.3). These base extensions denote the set of possible non-empty intersections of the input class extensions. For instance, the base extension 1 contains all objects which are only members of the class $P3$, whereas the base extension 6 consists of objects which are members of the four classes $P1$, $P2$, $P3$, and M at the same time.

Base Ext.	1	2	3	4	5	6	7	8	9
P1	0	1	1	1	1	1	1	0	0
P2	0	0	0	1	1	1	1	1	1
M	0	0	0	0	0	1	1	1	0
P3	1	1	0	0	1	1	0	0	0

Figure 2.3: Base Extensions of the Running Example

In detail, in our running example the following binary extensional relationships exist: $P1$ intersects with $P3$, $P1$ intersects with $P2$, $P1$ intersects with M, $P3$ intersects with $P2$, $P3$ intersects with M, and M is subset of $P1$. The resulting schema of an upward inheritance based on the given binary relationships is depicted in Figure 2.4.

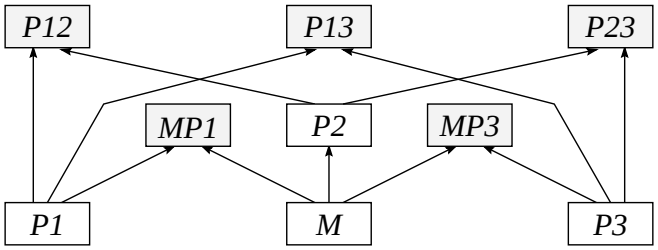

Figure 2.4: Integrated Schema by Upward Inheritance

If we now compare the set of base extensions with the result of the upward inheritance, we see that base extension 6 is part of all classes of the integrated schema, i.e., there is no

most specialized class which is the origin of this base extension. On the other hand, there is no most general class which contains all product objects. These objects are stored redundantly distributed over several classes. Indeed, the result of the upward inheritance is not false with respect to the origin schemata, but it does not exactly reflect the origin relationships. Strictly speaking, the result of the integration by upward inheritance is no real integrated schema. Rather it is a merged schema where the existing redundancy is not solved. However, this result is not astonishing since the integration methods based on upward inheritance does not consider all possible (n-ary) extensional relationships. For instance, in the example above the information that the intersection of the classes $P2$ and $P3$ is a subset of $P1$ is not considered. Please note that upward inheritance can be used *recursively* to derive classes from derived classes, e.g. to derive the class $P123$ from the classes $P12$ and $P23$. As a result, a recursive usage of upward inheritance may lead to an explosion of the number of derived classes.

Therefore, we proposed an integration method [SS96b, SS96a, SS97] basing on a *base extension set* which is determined by the extensional relationships obtained during the extension analysis. Due to the fact that the designer often cannot overview all existing (relevant) extensional relationships of the universe of discourse at once, we suggest an incremental integration where the set of base extensions given by the set of assertions is incrementally refined by changing the set of assertions.

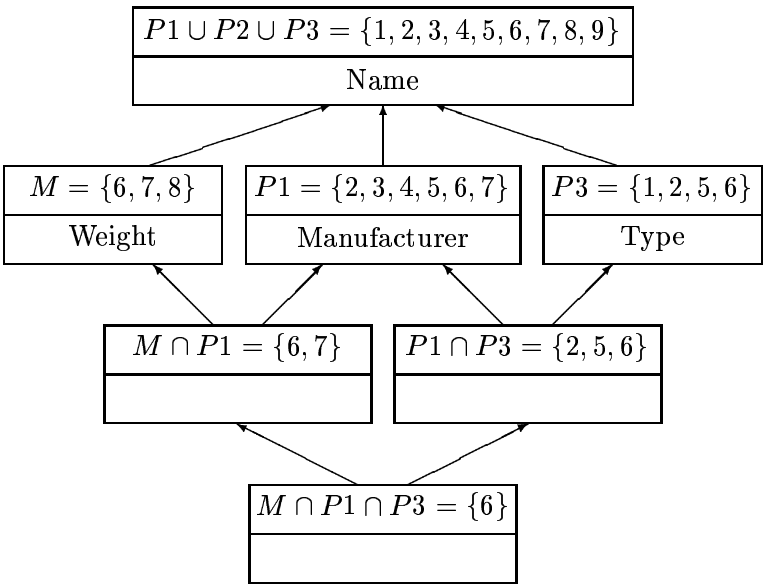

Figure 2.5: Complete Integrated Schema

The correct and complete integrated schema according to the given correct and complete set of base extensions is depicted in Figure 2.5. This integrated schema derived by the algorithm described in [SS97]. The root class contains all objects of the universe of discourse (union of all base extensions). These objects have the attribute 'Name' in common. The root class has three subclasses M, $P1$, and $P3$ which contain the same

objects as the original classes with these names. The further specialized classes $M \cap P1$, $P1 \cap P3$, and $M \cap P1 \cap P3$ have no additional attributes. These classes are generated in order to maintain the original semantics of the schemata to be integrated. The class $M \cap P1 \cap P3$, for instance, corresponds to the base extension 6 and, thus, contains all objects which are members of all classes to be integrated.

Chapter 3

Basic Notions and Concepts

An *object* models an entity of the universe of discourse. Objects with similar properties are grouped into *classes*, i.e., a class contains a set of objects. We assume that each object is uniquely identified by its *object identifier*.

Definition 3.1 (State of a Class) The *state* of a class C at time t, denoted as $State_C^t$, is the set of its objects at that time. Often, the state of a class is also called (class) extension. Please note that $State_C^t$ can also refer to a set expression defined over several classes. □

Extensional relationships between classes are determined by the corresponding possible states.

Definition 3.2 (Extensional Relationship) For two classes A and B we define five possible (binary) extensional relationships. These relationships are expressed by assertions which have to be always valid:

$$\begin{aligned}
\textit{Equivalence}\colon A \equiv B &\;:\Leftrightarrow\; \forall t\colon State_A^t = State_B^t \\
\textit{Inclusion}\colon A \subset B &\;:\Leftrightarrow\; \forall t\colon State_A^t \subseteq State_B^t \\
\textit{Containment}\colon A \supset B &\;:\Leftrightarrow\; \forall t\colon State_A^t \supseteq State_B^t \\
\textit{Disjointness}\colon A \varnothing B &\;:\Leftrightarrow\; \forall t\colon State_A^t \cap State_B^t = \varnothing \\
\textit{Overlap}\colon A \Cap B &\;:\Leftrightarrow\; \neg(A \equiv B) \wedge \neg(A \subset B) \wedge \neg(A \supset B) \wedge \neg(A \varnothing B)
\end{aligned}$$

Please note that the definitions above base on the notion of semantical equivalence, i.e., two objects may correspond to the same real-world object although they have different attributes or belong to different classes. In this way, we say two classes are *extensionally equivalent* if and only if their extensions always correspond to the same set of real world objects. The other assertions have to be read analogously. □

However, with binary assertions we cannot get a grasp of all possible relationships among a set of classes [SS96b]. For expressing n-ary extensional relationships among several classes we additionally need set operations.

Definition 3.3 (Set Operation) For three classes A, B, and C we define the following set operations:

$$\begin{aligned} \textit{Difference}: \quad & A \setminus B = C \quad :\Leftrightarrow \quad State_A \setminus State_B = State_C \\ \textit{Intersection}: \quad & A \cap B = C \quad :\Leftrightarrow \quad State_A \cap State_B = State_C \\ \textit{Union}: \quad & A \cup B = C \quad :\Leftrightarrow \quad State_A \cup State_B = State_C \end{aligned}$$

Please note that a class in an assertion can be substituted by a set expression consisting of $n+1$ classes and n set operations. □

For example, we are now able to define that the intersection of the classes A and B should always contain the difference between the classes C and D by the following expression:

$$(A \cap B) \supset (C \setminus D)$$

In summary, the complete grammar of the assertion language used here is as follows:

```
<assertion> ::= <expr> <rel-op> <expr>
<expr>      ::= (<expr>)
                | <class-id>
                | <expr> <set-op> <expr>
<rel-op>    ::= ≡ | ⊂ | ⊃ | ∅ | ⫛
<set-op>    ::= \ | ∩ | ∪
```

Our assertion language enables also the definition of *prohibitions*. For instance, we can define that the difference between the class A and the union of the classes B and C must be always empty:

$$(A \setminus (B \cup B)) \equiv (A \setminus A)$$

In the following, we will use the symbol '$\emptyset$' (instead of writing e.g. $(A \setminus A)$) to denote an empty set.

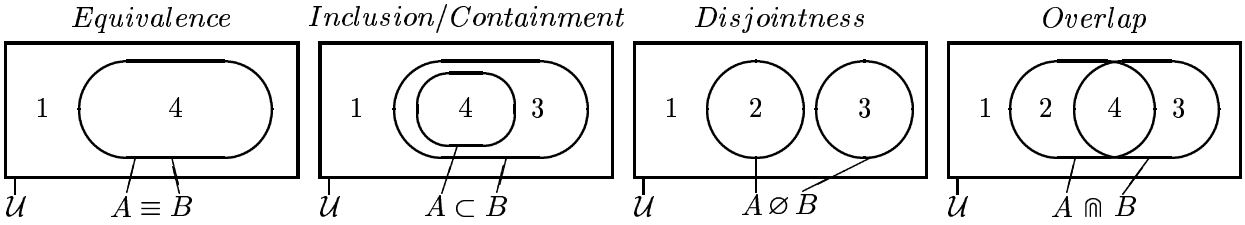

Figure 3.1: All Possible Extensional Relationships between Two Classes A and B

Figure 3.1 graphically represents all possible (binary) extensional relationships between two classes A and B. Depending on the relationship, the database universe $\mathcal{U}$ is decomposed into several sets of states, the so-called *base extensions* (abbreviated by the numbers in the brackets). In case of two classes A and B there are maximal four base extensions:

$$[1]\ \mathcal{U} \setminus (A \cup B),\ [2]\ A \setminus B,\ [3]\ B \setminus A, \text{ and } [4]\ A \cap B.$$

As depicted in Figure 3.2, the (binary) relationships as well as the (binary) set operations can be easily expressed by boolean formulas in disjunctive (DNF) or conjunctive normal form (CNF)[1].

	DNF	CNF
$A \equiv B$	$A\,B + \overline{A}\,\overline{B}$	$(A + \overline{B})(\overline{A} + B)$
$A \subset B$	$A\,B + \overline{A}\,B + \overline{A}\,\overline{B}$	$\overline{A} + B$
$A \supset B$	$A\,B + A\,\overline{B} + \overline{A}\,\overline{B}$	$A + \overline{B}$
$A \oslash B$	$A\,\overline{B} + \overline{A}\,B + \overline{A}\,\overline{B}$	$\overline{A} + \overline{B}$
$A \Subset B$	$A\,B + A\,\overline{B} + \overline{A}\,B + \overline{A}\,\overline{B}$	$true$
$A \setminus B$	$A\,\overline{B}$	$A\,\overline{B}$
$A \cap B$	$A\,B$	$A\,B$
$A \cup B$	$A\,B + A\,\overline{B} + \overline{A}\,B$	$A + B$

Figure 3.2: Binary Assertions and Set Operations as Boolean Formulas in DNF and CNF

N-ary assertions can also be transformed into a boolean formula in DNF or CNF. Below we show the transformation of the (n-ary) assertion above into the disjunctive normal form:

$$(A \cap B) \supset (C \setminus D) \rightsquigarrow (A\,B + \overline{C\,\overline{D}}) \rightsquigarrow (A\,B + \overline{C} + D)$$

Definition 3.4 (Base Extension Set) The *base extension set* $BES_{A_1,...,A_n}$ of the classes $A_1, ..., A_n$ is defined by a boolean formula in disjunctive canonical form basing on the variables A_1, ..., A_n. A minterm [Bis93] in such a formula denotes a *base extension*. Each possible object of the classes A_1, ..., A_n can be assigned to exactly one base extension. □

Thus, there are maximal 2^n possible base extensions for n related classes. For example, if there are two classes A and B and a relationship $A \Subset B$ in the universe of discourse, then there are four base extensions which are expressed by the following minterms:

$$A\,B,\ A\,\overline{B},\ \overline{A}\,B, \text{ and } \overline{A}\,\overline{B}.$$

The minterm $A\,\overline{B}$, for instance, stands for the base extension containing all objects of class A which are not also instances of class B.

[1] The boolean operations conjunction and disjunction are denoted by $\cdot$ and $+$, respectively. As a rule the $\cdot$ is often omitted.

Chapter 4

Incremental Computation of a Base Extension Set

A base extension set is essential for a correct schema integration. In [SS97] we presented an algorithm to derive an integrated schema from different schemata and a corresponding base extension set. The ideas behind this algorithm are presented in [SS96a]. In the following, we apply this algorithm without any further explanation.

This section describes how to compute a base extension set from a given set of assertions. The base extension set satisfies the constraints determined by the assertions. We will formally define the notion of satisfaction later.

During the integration process there are two sources of assertions. Assertions can be derived from schema definitions. A specialization relationship between two classes, for instance, means an extensional inclusion relationship. The other source of assertions is the database designer. The designer must have knowledge about the semantics of the schemata to be integrated in order to be able to specify relationships among classes of different schemata.

Suppose for n classes the set $\mathcal{A}$ contains all[1] possible assertions and the set $\mathcal{M}$ contains all corresponding 2^n minterms. A base extension set is defined by $BES \in 2^{\mathcal{M}}$. In the following, we develop the function *generate* which computes a base extension set from a given set of assertions:

$$generate : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{M}}$$

The complete definition of assertions *at one time* is a hard and often nearly impossible task for the designer. Usually the designer can give ad hoc only some assertions. In our approach, the base extension set and the integrated schema can be computed although the assertions are not complete. The existence of an integrated schema aids the designer in detecting new assertions as well as errors in the specified assertions. The new, improved set of assertions results then in a refined integrated schema. In this way, we speak

[1] $\mathcal{A}$ also includes prohibitions as a special form of assertions.

of an incremental integration process. If the assertions are complete and correct, then the derived base extension set is also correct. From a correct base extension set results a correct integrated schema. The incremental process[2] is illustrated in Figure 4.1.

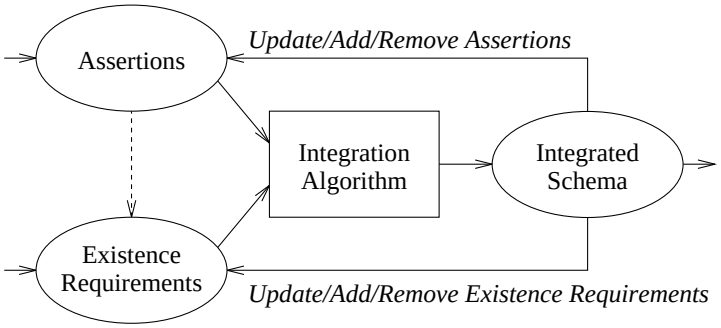

Figure 4.1: The Incremental Integration Process

Our example schemata define a specialization relationship between the classes M and $P2$. Figure 4.2 depicts the corresponding assertion and its transformation into a boolean formula in CNF. Additionally, the designer specifies some binary assertions which are presented in Figure 4.3.

Assertion	CNF	Existence
$M \subset P2$	$\overline{M} + P2$	$M\ P2, \overline{M}\ P2$

Figure 4.2: Assertions from the Schemata

Assertion	CNF	Existence
$P1 \Cap P2$	true	$P1\ P2, P1\ \overline{P2}, \overline{P1}\ P2$
$P1 \Cap M$	true	$P1\ M, P1\ \overline{M}, \overline{P1}\ M$
$P1 \Cap P3$	true	$P1\ P3, P1\ \overline{P3}, \overline{P1}\ P3$

Figure 4.3: Binary Assertions from the Designer

Based on these incomplete assertions the base extension set must be computed.

A straightforward, naive approach to implement the function *generate* basing on a given set of assertions $A = \{a^1, \ldots, a^m\}$ over n classes is described in the following:

1. Transform each assertion $a \in A$ into a boolean formula in disjunctive normal form, denoted by 'a_{DNF}'.

[2]The notion of an existence requirement will be explained later.

2. Combine all assertions conjunctively into one logical expression: $a^1_{DNF}\, a^2_{DNF} \ldots a^m_{DNF}$.

3. Transform the combined logical expression into the disjunctive canonical form over n classes. The result represents the base extension set.

The problem of this approach is the explosion of the number of base extensions. If, for instance, for 10 classes only the assertion $A \subset B$ is specified, then it forbids base extensions containing the term $A\overline{B}$ from all possible base extensions. The algorithm computes $\frac{3*2^{10}}{4} = 768$ base extensions. Of course, he exponential complexity is too high for practical use.

The base extension set of Figure 4.4 demonstrates the existence of a base extension set with less base extensions. It satisfies the given assertion and the *default assertion* which states that for each class there must exist at least one base extension.

Base Ext.	1	2	3	4	5	6	7	8	9	10
A	1	0	0	0	0	0	0	0	0	0
B	1	1	0	0	0	0	0	0	0	0
C	0	0	1	0	0	0	0	0	0	0
D	0	0	0	1	0	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0	0
F	0	0	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	1	0	0	0
H	0	0	0	0	0	0	0	1	0	0
I	0	0	0	0	0	0	0	0	1	0
J	0	0	0	0	0	0	0	0	0	1

Figure 4.4: Base Extension Set Fulfilling the Assertions

The base extension set in Figure 4.4 is only one example of a base extension set satisfying the given assertions. Typically, more than one satisfying base extension set can exist. In this section, we try to find a base extension set with a small number of base extensions. This base extension set, furthermore, has to be computable in a practicable time.

The main idea of our approach is the *simplification of logical expressions in an incremental integration process.*

In order to compute the base extension set for a given assertion set $A = a^1, \ldots, a^m$, we suggest the application of logical simplification laws as follows:

1. Transform each assertion $a \in A$ into a boolean formula in conjunctive normal form, denoted by 'a_{CNF}'.

2. Combine all assertions conjunctively into one logical expression: $a^1_{CNF}\, a^2_{CNF} \ldots a^m_{CNF}$.

3. Transform the combined logical expression into the disjunctive normal form. Do this by applying the following simplification laws after each application of the distributive law:

$$\begin{aligned}
\textit{Idempotence:} \quad A + A &= A \\
A\,A &= A \\
\textit{Complement:} \quad A + \overline{A} &= true \\
A\,\overline{A} &= false \\
\textit{Absorption:} A + (A\,B) &= A \\
A\,(A + B) &= A \\
\textit{Identity:} \quad A + true &= true \\
A + false &= A \\
A\,true &= A \\
A\,false &= false
\end{aligned}$$

The result is a logical expression in disjunctive normal form, which is, however, not in the disjunctive canonical form. Hence, it is not a complete base extension set. The expression has a relatively small number of terms and is logically correct. However, the expression does not satisfy the full semantics of the assertions. Additional steps must follow in order to compute a complete base extension set.

The application of the proposed three steps to our example assertions (cf. Figures 4.2 and 4.3) produces the following boolean formula:

$$\overline{M} + P2.$$

As we will describe in the next paragraph, this formula does not meet the semantics of the assertions completely.

In addition to the logical semantics an assertion states existence requirements. The assertion $A \Cap B$, for instance, requires the existence of the terms $A\,B$, $A\,\overline{B}$, and $\overline{A}\,B$ in the base extension set. In other words, if the classes A and B overlap, then objects can exist, which are instances of both classes simultaneously, and objects may occur, which are instances of only one class.

Definition 4.1 (Existence Requirement) An *existence requirement* e is a logical expression which must be expressed by at least one base extension of the BES. If $State_e^t$ denotes the State of e (as a set expression) at the time t, then the existence requirements means:

$$\exists t : State_e^t \neq \emptyset$$

□

Figure 4.5 shows the existence requirements for each binary assertion. The existence requirements of our example form the third column in the Figures 4.2 and 4.3.

Assertion	Existence
$A \equiv B$	$A\,B$
$A \subset B$	$A\,B, \overline{A}\,B$
$A \supset B$	$A\,B, A\,\overline{B}$
$A \varnothing B$	$A\,\overline{B}, \overline{A}\,B$
$A \Subset B$	$A\,B, A\,\overline{B}, \overline{A}\,B$

Figure 4.5: Existence Requirements

In addition to existence requirements stemming from assertions, independent assertions are needed. One kind of an independent assertion is the *default existence requirement* which demands that there must exist at least one base extension for each class. Furthermore, the designer often wants to define existence requirements independently from assertions. In this way, he can express the existence of a non-empty set expression over a given set of classes.

Besides the specified assertions, the existence requirements must also be considered for the computation of a base extension set. Suppose, the set E contains the existence requirements stemming from the given assertions and additional independent existence requirements as boolean formulas in disjunctive normal form. The following definition fixes the satisfaction of assertions and existence requirements by a base extension set.

Definition 4.2 (Satisfaction by a BES) Suppose, the set $A = \{a^1_{CNF}, \ldots, a^k_{CNF}\}$ denotes the set of assertions in conjunctive normal form and $E = \{e^1_{DNF}, \ldots, e^l_{DNF}\}$ the set of existence requirements in disjunctive normal form for n classes. An existence requirement $e^i_{DNF} \in E$ consists of a set of terms $et^{i_1}_{DNF}, \ldots, et^{i_p}_{DNF}$.

A base extension set in disjunctive canonical form $BES = \{m^1, \ldots, m^m\}$ with $m^i = v^{i_1} \ldots v^{i_n}$ where each variable (class) v^{i_j} is either negated ($v^{i_j}_-$) or not ($v^{i_j}_+$) *satisfies* A and E if:

- All assertions of A are satisfied by the base extension set:

$$\forall a_{CNF} \in A : BES \Rightarrow a_{CNF}$$

- All existence requirements of E are satisfied by the base extension set:

$$\forall e_{DNF} \in E : \exists et_{DNF} \in e_{DNF} : \exists m \in BES : m \Rightarrow et_{DNF}$$

□

As motivated above, the function *generate* has to consider existence requirements besides assertions in CNF. We denote all possible existence requirements over n classes by $\mathcal{E}$.

The input of the function must be expanded:

$$generate : (2^{\mathcal{A}} \times 2^{\mathcal{E}}) \rightarrow 2^{\mathcal{M}}$$

In order to consider assertions (denoted by A) and existence requirements (denoted by E), we extend the proposed three steps by further steps:

4. Transform the result of step 3 to a so-called *source table*. In that table each row corresponds to a class (variable) and each column to a term of the simplified formula (result of step 3). For each non-negated variable (class) in a term write '1' into the corresponding field. A negated variable is represented by a '0'. A missing variable in a term corresponds to an empty field.

   Applying the first three steps to our example provides $P2 + \overline{M}$. Figure 4.6 shows its transformation to a source table.

P1		
P2	1	
M		0
P3		

Figure 4.6: Source Table of the Example

5. Create an empty *target table* with n rows for the n classes and no column.
6. Perform the following steps for each $e \in E$:
   (a) If the target table already satisfies e then continue with step 6.
   (b) If e can be satisfied by the target table filling empty fields with '0' or '1', then fill those values into the empty fields of the first suitable column of the target table. Fill in only values which are needed for the satisfaction. Continue with step 6.
   (c) Take over the first suitable column from the source to the target table and fill, if necessary for satisfaction, empty fields with corresponding values as described in the previous step.
7. Fill up the remaining empty fields of the target table with '0'.

The complexity of the steps above is polynomial depending on the number of existence requirements and the number of columns of the source table.

In the introduced example following existence requirements are derived from the assertions:

$$E = \{P1\,P2, P1\,\overline{P2}, \overline{P1}\,P2, P1\,M, P1\,\overline{M}, \overline{P1}\,M,$$
$$P1\,P3, P1\,\overline{P3}, \overline{P1}\,P3, M\,P2, \overline{M}\,P2\}$$

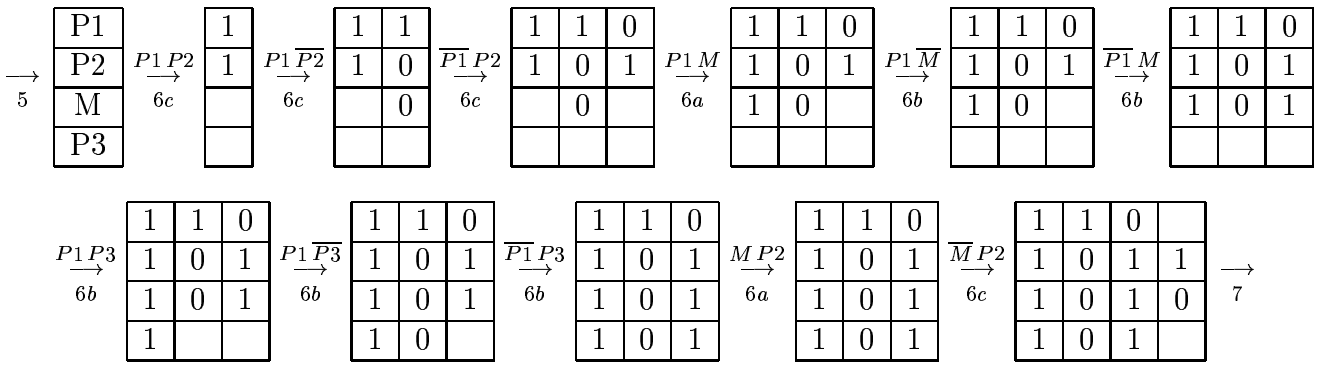

Figure 4.7: Evolution of the Target Table during Step 6

The application of steps 4–7 using the existence requirements (in the presented order) and the source table in Figure 4.6 produces the base extension set in Figure 4.8. Step 6 for the example is sketched in Figure 4.7.

The first existence requirement $P1\,P2$ could not be satisfied by the empty target table (steps 6a and 6b). Therefore, the first column of the source table is taken over to the target table. Filling '1' in row $P1$ satisfies the existence requirement (step 6c). Continuing step 6 (see also Figure 4.7) and 7 for the remaining existence requirements produces the base extension set depicted in Figure 4.8. The base extensions are numbered corresponding to the column numbers of the BES in Figure 2.3. The third column has no correspondence and is therefore identified by '$x1$'.

Base Ext.	6	3	x1	9
P1	1	1	0	0
P2	1	0	1	1
M	1	0	1	0
P3	1	0	1	0

Figure 4.8: Base Extension Set

Figure 4.9 shows the extensional relationships graphically. The integration algorithm presented in [SS97] generates the corresponding integrated schema (cf. Figure 4.10).

Following the steps 1–7 of the proposed algorithm, we distinguish four different states of the integration process depending on the given set of assertions A and the set of existence requirements E:

1. The base extension set cannot be computed since the assertions are conflicting. If, for instance, $A \oslash B$ and $A \subset B$ are defined at the same time then the existence requirement $A\,\overline{B}$ derived from $A \oslash B$ cannot be satisfied by the corresponding source table. In general, in this state the source table cannot satisfies all existence requirements.

2. The base extension set can be computed. However, it does not represent the correct

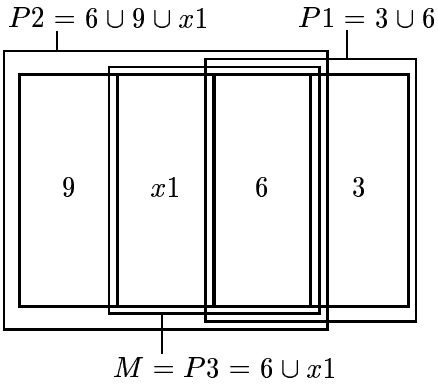

Figure 4.9: Base Extension Set as Diagram

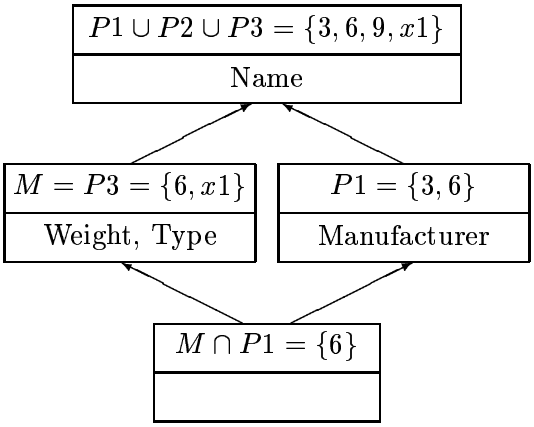

Figure 4.10: Integrated Schema

real-world semantics. Different orders of existence requirements provide different base extension sets. Depending on completeness of A and E we distinguish two further sub-states:

(a) Wrong base extensions can occur because the set A of assertions is incomplete or wrong.

(b) The set A of assertions is complete and correct. However, the set E is incomplete.

3. The computed base extension set represents the correct real-world semantics since A and E are complete and correct.

The incremental process of schema integration is characterized by going from states 1 and 2a to the states 2b and 3. The state 3 indicates the end of the incremental process. For the transition from state 2a to state 3 many existence requirements have to be added to E. This transition step can be simplified by transforming the source table logically into disjunctive canonical form. Steps 4–7 can then be omitted. The problem, however,

is to detect whether a process is in state 2b. If a process is not in state 2b and the source table is transformed into disjunctive canonical form, then the resulting set of base extensions can explode. An estimation of the number of base extension basing on empty fields in the source table helps to overcome that problem. If the estimated number of base extensions is less than a given upper bound, then the transformation is practicable and no further existence requirement is needed. The designer can set the value of the upper bound.

In the states 2a and 2b different orders of existence requirements result in different base extension sets and consequently in different integrated schemata. The designer can force another integrated schema by changing the order of the existence requirements to find more errors or hints for further assertions.

Now we come back to our example. The integration process is in state 2a. Base extension $x1$ is wrong because this set is always empty. Furthermore, not all assertions are specified. The integrated schema (cf. Figure 4.10) shows the extensional equivalence of the classes M and $P3$ which is not correct. Furthermore, the BES express $P3$ as a subset of $P2$ which is wrong, too. We add therefore the assertions $P2 \Cap P3$ and $M \Cap P3$ to the set of assertions (cf. Figure 4.11).

Assertion	CNF	Existence
$P1 \Cap P2$	true	$P1\,P2, P1\,\overline{P2}, \overline{P1}\,P2$
$P1 \Cap M$	true	$P1\,M, P1\,\overline{M}, \overline{P1}\,M$
$P1 \Cap P3$	true	$P1\,P3, P1\,\overline{P3}, \overline{P1}\,P3$
$P2 \Cap P3$	true	$P2\,P3, P2\,\overline{P3}, \overline{P2}\,P3$
$M \Cap P3$	true	$M\,P3, M\,\overline{P3}, \overline{M}\,P3$

Figure 4.11: Complete Binary Assertions from the Designer

Applying the first four steps to the extended set of assertions yields the same source table as before (cf. Figure 4.6). The following existence requirements are derived from the assertions:

$$\begin{aligned} E = \{&M\,P2, \overline{M}\,P2, P1\,P2, P1\,\overline{P2}, \overline{P1}\,P2, \\ &P1\,M, P1\,\overline{M}, \overline{P1}\,M, P1\,P3, P1\,\overline{P3}, \overline{P1}\,P3, \\ &P2\,P3, P2\,\overline{P3}, \overline{P2}\,P3, M\,P3, M\,\overline{P3}, \overline{M}\,P3\} \end{aligned}$$

Steps 5–7 produces the base extension set presented in Figure 4.12. Figure 4.13 shows the same information graphically. The corresponding schema is depicted in Figure 4.14.

Due to the wrong base extension $x2$, the process is still in state 2a. The assertions are still incomplete. Each class, however, is compared with each other and forms an existing binary assertion. No additional binary assertion to the specified ones can exist. A further assertion is needed which is not a binary assertion. The needed n-ary assertion

Base Ext.	6	x2	3	8	1
P1	1	0	1	0	0
P2	1	1	0	1	0
M	1	0	0	1	0
P3	1	1	0	0	1

Figure 4.12: Base Extension Set from all Binary Assertions

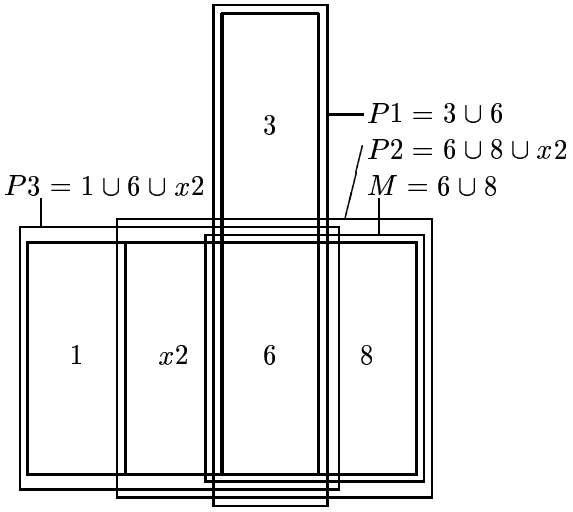

Figure 4.13: Base Extension Set as a Diagram

express that the intersection of the classes $P2$ and $P3$ is always a subset of class $P1$ (cf. Figure 4.15). This assertion forbids the existence of the base extension $x2$.

Steps 1–3 using all known assertions produces following formula:

$$(\overline{M} + P2)(P1 + \overline{P2} + \overline{P3}) = P1\,\overline{M} + \overline{P2}\,\overline{M} + \overline{M}\,\overline{P3} + P1\,P2 + P2\,\overline{P3}$$

The corresponding source table is depicted in Figure 4.16.

The set of assertions is complete because the transformation of the source table into the disjunctive canonical form would produce the correct base extension set (cf. Figure 2.3). The derivation of the base extension set using steps 6 and 7 provides an incomplete base extension set (cf. Figure 4.17), because the set of existence requirements is not complete.

Figure 4.18 shows the base extension set graphically whereas Figure 4.19 depicts the resulting integrated schema.

The integration process is now in state 2b. The search for the missing existence requirements can be omitted by transforming the source table into disjunctive canonical form. The resulting base extension set is then correct and leads to the correct integrated schema (cf. Figure 2.5).

A difference between the schemata depicted in Figures 4.19 and 2.5 is the existence of the class '$M \cap P1$' containing objects which are instances of the classes M and $P1$

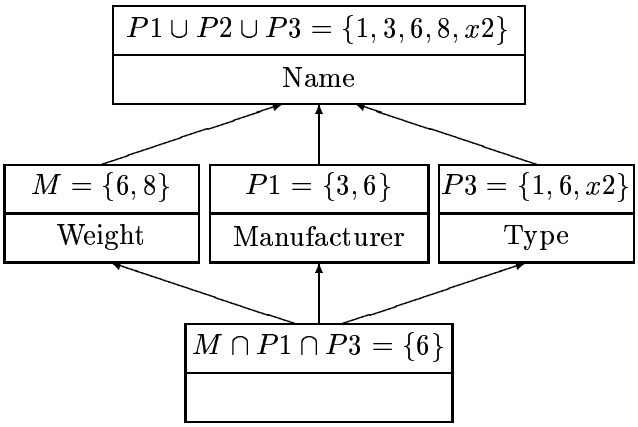

Figure 4.14: Refined Integrated Schema Reflecting all Binary Assertions

Assertion	CNF	Existence
$(P2 \cap P3) \subset P1$	$P1 + \overline{P2} + \overline{P3}$	$P1\,P2\,P3$, $P1\,\overline{P2} + P1\,\overline{P3}$

Figure 4.15: N-ary Assertion from the Designer

but not of the class $P3$ (base extension 7). This existence requirement was not specified and is therefore not considered in Figure 4.19.

In the used example we compared the intermediate integrated schemata with the given correct schema depicted in Figure 2.5. Obviously, in a practical environment the designer has to refer to his knowledge about the semantics of the schemata to be integrated instead to a correct integrated schema. In this way the designer incrementally approaches to the correct integrated schema.

P1	1			1	
P2		0		1	1
M	0	0	0		
P3			0		0

Figure 4.16: Complete Source Table of the Example

Base Ext.	1	3	4	5	6	8	9
P1	0	1	1	1	1	0	0
P2	0	0	1	1	1	1	1
M	0	0	0	0	1	1	0
P3	1	0	0	1	1	0	0

Figure 4.17: Base Extension Set from all Assertions including Existence Requirements

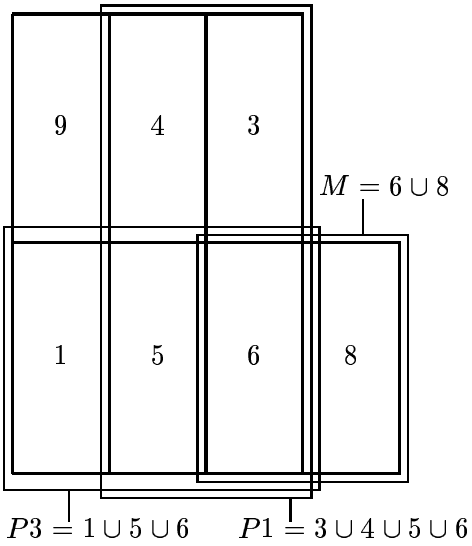

Figure 4.18: Refined Base Extension Set as Diagram

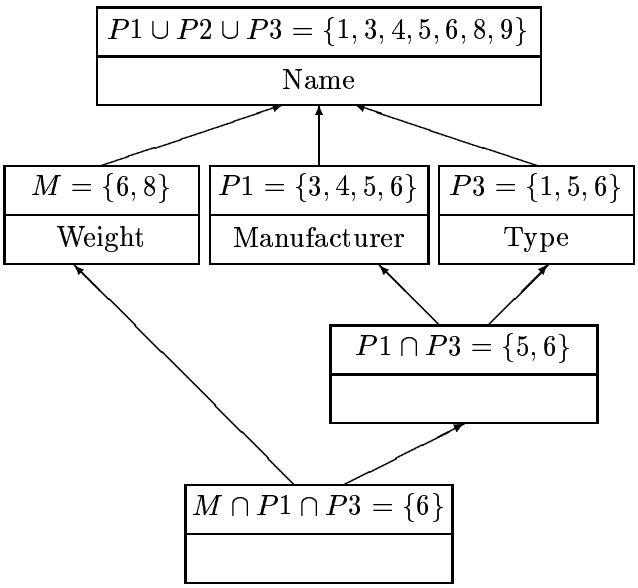

Figure 4.19: Refined Integrated Schema

Chapter 5

Conclusions

In this paper, we emphasized the importance of considering precise extensional relationships as input for a correct and complete schema integration. We presented an approach to schema integration which supports an incremental refinement of the integrated schema. The quality of the integrated schema can be increased by enhancing the quality of the input information, i.e., by capturing additional or more precise information about the extensional relationships among the input object classes to be integrated. Furthermore, we pointed out that our approach supports the specification of all possible binary as well as n-ary extensional relationships. These relationships are formulated in form of assertions and can be logically represented by boolean formulas. These formulas restrict the database universe to the object classes which are mutually consistent with respect to the extensional relationships. We developed a method for computing the a base extension set from a given set of assertions. This base extension set is the input for our integration algorithm. We stressed that defining extensional relationships as boolean formulas alone is not enough to get an integrated schema which precisely represents the real-world. For that, existence requirements must also be considered.

The presented approach is realized as part of an interactive database integration prototype which aids the designer in performing a semantically correct integration. Our prototype was presented at CeBIT fair last year. Our future work will focus on exploiting the correspondence between integrity constraints and extensional relationships for schema integration.

Bibliography

[BC86] J. Biskup and B. Convent. A Formal View Integration Method. In C. Zaniolo, editor, *Proc. of the 1986 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C.*, pages 398–407, ACM SIGMOD Record, Vol. 15, No. 2, ACM Press, June 1986.

[Bis93] N. N. Biswas. *Logic Design Theory.* Prentice Hall, Eaglewoods Cliffs, NJ, 1993.

[BL84] C. Batini and M. Lenzerini. A Methodology for Data Schema Integration in the Entity-Relationship Model. *IEEE Transaction on Software Engineering*, 10(6):650–664, November 1984.

[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.

[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. International Society for Computers and Their Application, Six Forks Road, Releigh, NC, 1996.

[GCS95] 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.

[GSC96] M. Garcia-Solaco, F. Saltor, and M. Castellanos. Semantic Heterogeneity in Multidatabase Systems. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications*, chapter 5, pages 129–202, Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[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.

[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.

[NEL86] S. B. Navathe, R. Elmasri, and J. A. Larson. Integrating User Views in Database Design. *IEEE Computer*, 19(1):50–62, January 1986.

[NG82] S. B. Navathe and S. G. Gadgil. A Methodology for View Integration in Logical Database Design. In D. MacLeod and Y. Villasenor, editors, *Proc. of the 8th Int. Conf. on Very Large Data Bases (VLDB'82), Mexico City*, pages 142–164, Morgan Kaufmann Publishers, September 1982.

[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.

[PBE95] E. Pitoura, O. Bukhres, and A. K. Elmagarmid. Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, 27(2):141–195, June 1995.

[RPRG94] 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.

[SK93] 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, Amsterdam, 1993.

[SL90] 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.

[SP94] S. Spaccapietra and C. Parent. View Integration: A Step Forward in Solving Structural Conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, April 1994.

[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*, Lecture Notes in Computer Science, Vol. 1021, pages 400–411. Springer-Verlag, Berlin, 1995.

[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*, Lecture Notes in Computer Science, Vol. 1157, pages 195–210. Springer-Verlag, Berlin, 1996.

[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. International Society for Computers and Their Application, Six Forks Road, Releigh, NC, 1996.

[SS97] I. Schmitt and G. Saake. Merging Inheritance Hierarchies for Schema Integration based on Concept Lattices. Preprint 2, Fakultät für Informatik, Universität Magdeburg, February 1997. Also available via http://wwwiti.cs.uni-magdeburg.de/publikationen/97/SS97.ps.gz.

[Wid95] J. Widom. Research Problems in Data Warehousing. In N. Pissinou, A. Silberschatz, E. K. Park, and K. Makki, editors, *Proc. of the 4th Conf. on Information and Knowledge Management (CIKM'95), Baltimore, Maryland, USA*, pages 25–30, ACM Press, November 1995.