Merging Inheritance Hierarchies for Schema Integration based on Concept Lattices [1]

Ingo Schmitt & Gunter Saake

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg
Germany

Email: {schmitt|saake}@iti.cs.uni-magdeburg.de

Tel.: ++49-391-67-12994
Fax: ++49-391-67-12020

Februar 1997

[1] This research was partially supported by the German Country Sachsen-Anhalt under FKZ: 1987A/0025 ("Föderierung heterogener Datenbanksysteme und lokaler Datenhaltungskomponenten zur systemübergreifenden Integritätssicherung").

Abstract

Merging inheritance hierarchies is an essential task of schema integration as part of the design of a federated database. Based on the formalization of inheritance hierarchies as concept lattices we present an efficient algorithm for deriving an integrated inheritance hierarchy as result of merging two inheritance hierarchies with overlapping extensions and types. Several methods are presented to modify the integrated hierarchy to optimize the resulting object-oriented schema with respect to certain quality criteria (disjoint specialization, number of classes, support of null values, etc). These methods are necessary to adjust the integrated schema to specific requirements and for realizing application-specific views.

Keywords: federated database design, schema integration, inheritance hierarchy, concept lattices

Contents

Chapter 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. [SL90]) 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 by schema integration. In general, this step is a very complex task due to many kinds of conflicts. Therefore, different integration methodologies were proposed in [BLN86, LNE89, SPD92, Dup94, RPRG94, PBE95] which describe various conflict resolution techniques.

Especially the merging of different inheritance hierarchies, the focus of this work, results in inheritance hierarchies which do not sufficiently fulfill the demand for minimality and understandability (cf. [TS93, GCS95]). Most of the proposed integration methodologies resolve semantical overlappings by introducing generalized classes (upward inheritance), e.g. [RPRG94, GCS95]. That is, the original inheritance hierarchies are subhierarchies of the resulting merged hierarchy. Henceforth, the resulting hierarchy can become unnecessarily complex. We demonstrated that in [SS96b].

Most of the proposed schema integration techniques including view integration techniques (e.g. [MNE88, NEL86, SP94]) are based on a data model similar to the ER-model extended by subtype relationships. They take over the inheritance hierarchies from the local to the integrated schema level and adapt them to each other by adding new sub-or superclasses or deleting existing classes.

In contrast to these approaches, we decompose overlapping class extensions into base extensions (cf. [SS96b, SS96a]) and use algorithms known from the theory of concept lattices [Wil92]. The new aspect proposed in this work is that we combine this theory with schema integration and extensional decompositions.

We mean here with the *extension* of a class a set of *possible* objects. Such extensions are detected during the analysis phase within the schema integration process. An inheritance hierarchy defines subset relations between the classes appearing in this hierarchy. The *intension* of a class is the set of its attributes, i.e., the type of the objects in this class.

In this paper, we present a formalization of inheritance hierarchies using context information and concept lattices. We transform inheritance hierarchies to be integrated into one merged context. The theory of concept analysis offers an algorithm to generate from context information a concept lattice which can be regarded as an inheritance hierarchy. This algorithm, however, has an unacceptable complexity. Furthermore, not all derived concepts represent useful classes. Therefore, we propose an improved algorithm generating only useful classes in polynomial time.

Another kind of manipulation of inheritance hierarchies presented here is the transformation of a given inheritance hierarchy into another one. This is important for several steps during database design and maintenance:

- *View derivation* based on an existing conceptual schema. In the federated database design the derivation of external schemata from the integrated schema must be supported.
- *Schema optimization* based on given quality criteria for object-oriented database schemata.
- Transformation of an inheritance tree into an inheritance hierarchy and vice versa.

The last item is important as part of the object-oriented database design because most design models use multiple inheritance whereas implementation models often require an inheritance tree with simple inheritance.

The formalization and the algorithms are demonstrated using a small example scenario from a library application.

Chapter 2

Example Scenario

In order to demonstrate the formalization and the transformation algorithms, we use a simple example showing the integration of two databases. One database is a library database storing information about publications, where books and journal papers are special types of publications. This database has to be integrated with a project database which stores project publications but distinguishes more types of publications. In contrast to journal papers, we assume books and technical reports are non-refereed publications. `Book` and `Techn.-Rep.` are subclasses of the class `Non-Refereed`. The inheritance corresponding hierarchies of both databases are given in Fig. 2.1 and Fig. 2.2.

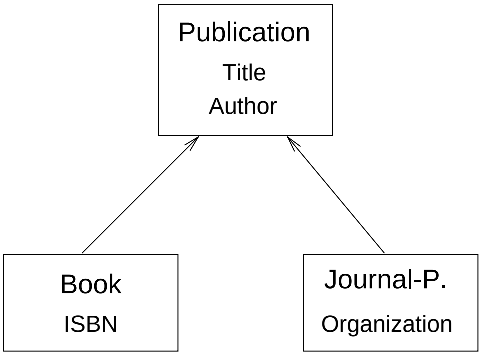

Figure 2.1: Library database schema

Before integrating both databases, we have to analyze the extensional overlappings between the classes to be integrated. Please notice that we mean with the *extension* of a class the set of *possible* objects. Information about extensional overlappings is the unavoidable basis for any integration of inheritance hierarchies. All approaches have to take extensional overlappings into consideration. This information has to be provided by interviewing database designers and by analyzing existing databases if available. The process of examining extensional overlappings can be very difficult. It can be made easier by using graphical representations, default assumptions, and methods to identify object isomerism from existing databases (cf. [CTK96]).

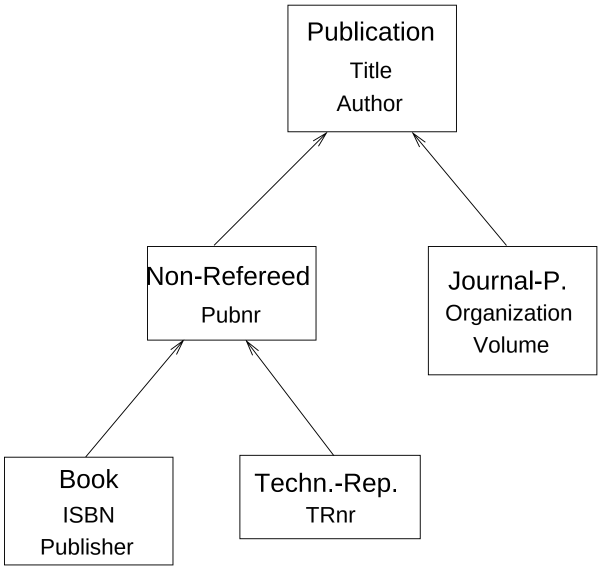

Figure 2.2: Project publication database schema

Let us assume for the example that we know about the extensional overlappings. For example, we know that `Publication` extensions of both databases overlap. The extensions of classes `Lib.Publication` and `Proj.Publication` represent the united extensions of their corresponding subclasses. Technical reports are stored in the `Project` database only. Some publications stored in the `Project` database are published simultaneously as a journal paper and a technical report.

We decompose extensional overlappings into base extensions (see Tab. 2.1). The original extensions are partitioned into base extensions which can be mapped to the input classes. For example, the `Book` extension of the library database is partitioned into the two base extensions `1` and `3`. The base extension `3` contains books from the library database which are also stored in the project database whereas the base extension `1` contains the remaining books.

Base extension	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Lib.Publication	$\surd$	$\surd$	$\surd$	$\surd$					
Lib.Book	$\surd$		$\surd$						
Lib.Journal-P.		$\surd$		$\surd$					
Proj.Publication			$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Proj.Journal-P.				$\surd$		$\surd$	$\surd$		
Proj.Non-Refereed			$\surd$		$\surd$		$\surd$	$\surd$	$\surd$
Proj.Book			$\surd$		$\surd$				
Proj.Techn.-Rep.							$\surd$	$\surd$	

Table 2.1: Extensional overlappings

We assume that attributes with same names have identical semantics. With other words, *attribute conflicts* [LNE89] are already resolved. The attribute-class relation is depicted in Tab. 2.2.

Attribute	**Title Author**	**ISBN**	**Organ.**	**Volume**	**Pub-lisher**	**TRnr**	**Pubnr**
Lib.Publication	√						
Lib.Book	√	√					
Lib.Journal-P.	√		√				
Proj.Publication	√						
Proj.Journal-P.	√		√	√			
Proj.Non-Refereed	√						√
Proj.Book	√	√			√		√
Proj.Techn.-Rep.	√					√	√

Table 2.2: Intensional overlappings

From extensional and intensional[1] information we are able to create a table, which assigns attributes to base extensions. A tick symbol for a column (base extension) and a row (attribute) means, that for possible objects of that base extension we would know the value of that attribute (Tab. 2.3).

Attr-Ext	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Title	√	√	√	√	√	√	√	√	√
Author	√	√	√	√	√	√	√	√	√
ISBN	√		√		√				
Organ.		√		√		√	√		
Volume				√		√	√		
Publisher			√		√				
TRnr							√	√	
Pubnr			√		√		√	√	√

Table 2.3: Extension-attribute relation

This table representation of the extensional and intensional aspects can directly be used for further analyses. The order of columns and rows is irrelevant: we can exchange rows and columns. With such exchanges we modify the representation only. In such a representation a rectangle corresponds to a class: a union of base extensions having the same set of defined attributes. An example of a class is formed by the set of base extensions {`3`, `5`} and by the set of attributes {`Publisher`, `Pubnr`}.

Moreover, we can detect subclass relationships using the rectangle representation of classes. If rectangles overlap as shown in Figure 2.3, then they are in a specialization relationship. For example the class with extension {`7`, `8`, `9`} and intension {`Pubnr`} is a superclass of the class with extension {`7`, `8`} and intension {`TRnr`, `Pubnr`}.

[1]The *intension* of a class is the set of its attributes.

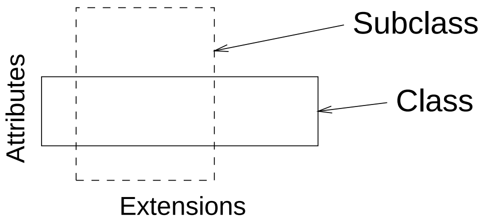

Figure 2.3: Class-subclass relation as overlapping rectangles

However, the detection of classes and subclass relationships using rectangles rely heavily on the chosen row and column order. Furthermore, to obtain an integrated schema meeting the requirement for minimality, maximal rectangles have to be found. Therefore, it is necessary to develop algorithms which perform this task independently of these sequences.

Chapter 3

Concept Lattices

In order to derive the integrated inheritance hierarchy we use the theory of concept lattices developed in the context of *formal context analysis* [Wil92]. Concept lattices, which are often called Galois lattices, are usually used in Artificial Intelligence to build conceptual classification.

The theory of concept lattices is based on the following formalization [Duq87]: A *context* (G, M, I) is given where G is a set of objects, M is a set of attributes (intension), and $I \subseteq G \times M$ is a binary relation between these (finite) sets, which expresses that the object $g \in G$ has the attribute $m \in M$ whenever $(g, m) \in I$ holds[1].

The *intent* of any object subset $A \subseteq G$ is defined by:

$$A' := \{m \in M \mid \forall g \in A : (g, m) \in I\}$$

and dually the *extent* of any set of attributes $B \subseteq M$ is defined by:

$$B' := \{g \in G \mid \forall m \in B : (g, m) \in I\}$$

A *concept* in (G, M, I) is a pair $(A, B) \in \mathcal{P}(G) \times \mathcal{P}(M)$ for which $A = B'$ and $B = A'$. It represents a maximal rectangle in the binary relation. Let

$$L := \{(A, B) \in \mathcal{P}(G) \times \mathcal{P}(M) \mid A = B' \wedge B = A'\}$$

be the set of all concepts (maximal rectangles or classes) in (G, M, I), and let $\leq$ be the order relation on L defined by:

$$(A_1, B_1) \leq (A_2, B_2) \Longleftrightarrow A_1 \subseteq A_2$$

As result, we have a lattice denoted by:

$$\mathcal{L} := (L, \leq, \wedge, \vee, (M', M), (G, G'))$$

[1] A context can be regarded as a table assigning base extensions to attributes.

The lattice operations are given by following definitions:

$$(A_1, B_1) \wedge (A_2, B_2) = (A_1 \cap A_2, (A_1 \cap A_2)')$$

and

$$(A_1, B_1) \vee (A_2, B_2) = ((B_1 \cap B_2)', B_1 \cap B_2).$$

The lattice built from concepts is called *concept lattice* [Duq87]. Elements of the lattice have both an extensional and an intensional aspect. This formal model can be adapted to the problem of schema integration. A lattice can be considered as an inheritance hierarchy as follows:

- objects of G correspond to base extensions;
- a concept in (G, M, I) is a class with extension and intension;
- $\leq$ is the specialization relationship between two classes;
- $\wedge$ is the specialization (intersection of extensions) of two classes;
- $\vee$ is the generalization (union of extensions) of two classes;
- (M', M) is the bottommost class of the hierarchy (intension is the union of all attributes; extension may be empty);
- (G, G') is the topmost class of the hierarchy (extension is the union of all base extensions; intension may be empty).

Based on this definition, we can directly build an inheritance hierarchy. Unfortunately, that approach has two disadvantages. If $\mid G \mid = m$ and $\mid M \mid = n$ then at most $2^{min(m,n)}$ concepts have to be be generated which results in an unacceptable complexity. An example of such a relation I is a matrix with n columns and n rows in which each matrix element has a tick except the matrix elements of the main diagonal. The lattice from this matrix contains 2^n concepts.

The second disadvantage is that some concepts represent unnecessary classes. Consider the relation shown in Table 3.1.

M/G	**1**	**2**	**3**	**4**
a			$\surd$	
b	$\surd$	$\surd$	$\surd$	
c		$\surd$	$\surd$	$\surd$
d		$\surd$		

Table 3.1: Relation producing unnecessary classes

The Hasse-diagram of the resulting concept lattice is depicted in Figure 3.1 (left diagram). The left diagram shows the concepts with the extensional and intensional

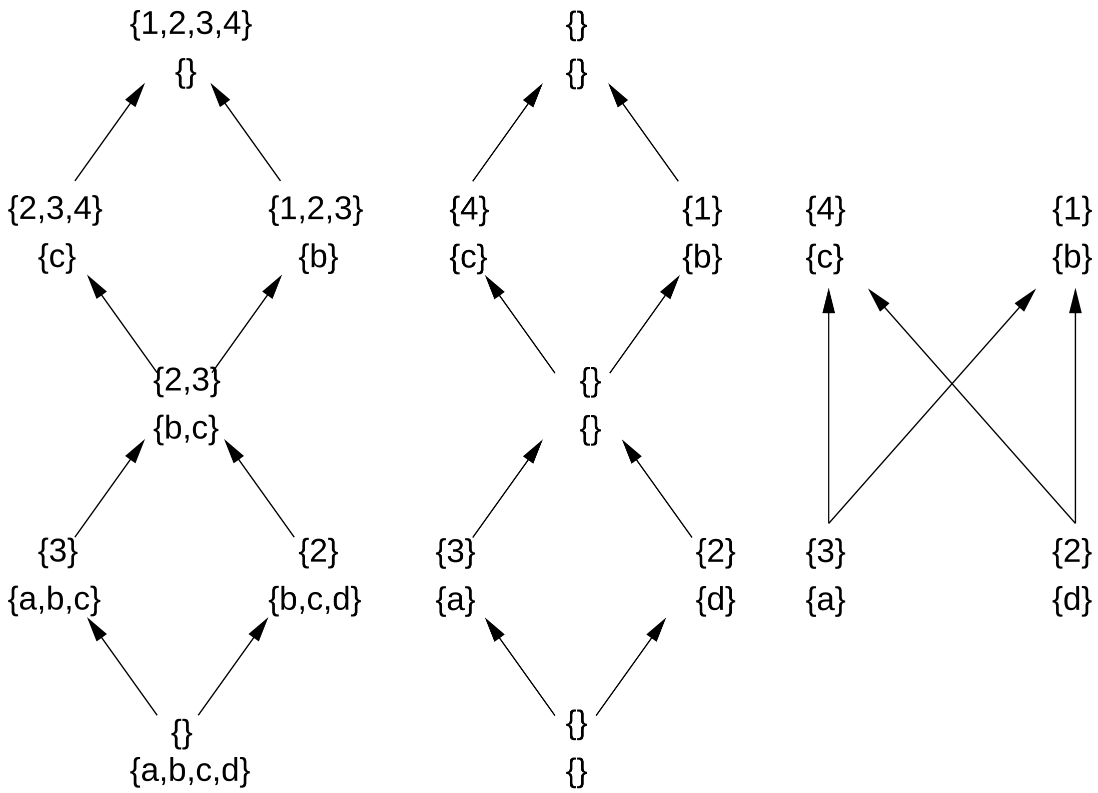

Figure 3.1: Resulting Galois lattices and inheritance hierarchy

elements, respectively. Due to the subset relationships between the intensions and the extensions, resp., of connected concepts the left diagram can be reduced to the middle diagram. This reduced diagram is only another representation of the same lattice. Intensions are inherited downwards and extensions are 'inherited' in the opposite direction. In the reduced diagram concepts with both empty intension and empty extension exist. If we regard these concepts as classes then they do not introduce attributes nor have extensions of their own. In other words, for these classes the intensions are given by the union of their superclass intensions and the extensions are given by the union of their subclass extensions. Classes without both intension and extension of their own can be omitted. Omitting such classes destroys the lattice property. In this case we use the term inheritance hierarchy instead of lattice. The right diagram of Figure 3.1 shows an inheritance hierarchie resulting from dropping classes without extensions and intensions of their own in the reduced representation.

We will refer to concepts (classes) having an extension or/and an intension of their own as *valid* concepts (classes). From the reduced Galois lattice it can easily be seen that at most $m+n$ valid classes can exist. In this case, each object (base extension) and each attribute is represented by a exactly one concept (class).

From the discussion exposes an interesting question: **Is there an algorithm to generate only valid concepts (classes) with polynomial time?**

Chapter 4

A Practicable Algorithm

In this section we present an algorithm which generates only valid classes with complexity $O(n^3)$. For each step of the following algorithm the complexity will be given. The input value for the complexity measurement is $n = max(|G|, |M|)$. G is the set of base extensions and M is the set of attributes. The complexity of a single step can be directly derived using the matrix representation as introduced in Section 2 (e.g. Table 2.3).

The sets Int_1 and Ext_1 contain the intents of each single base extension ($g \in G$) and the extents of each single attribute ($m \in M$), respectively:

$$Int_1 := \{\{g\}' \mid g \in G\} \qquad Ext_1 := \{\{m\}' \mid m \in M\}$$

The complexity to compute both sets is $O(n^2)$.

From Int_1 and Ext_1 two sets of classes are derived:

$$Con_{1,I} := \{(I', I) \mid I \in Int_1\} \qquad Con_{1,E} := \{(E, E') \mid E \in Ext_1\}$$

The complexity to derive both sets is $O(n^3)$. The set Int_1 contains at most n elements. Each element of Int_1 must be compared with each column (at most n columns) of the matrix. For each comparison at most n attributes have to be examined. The complexity computing $Con_{1,E}$ is analogous to $Con_{1,I}$.

We obtain a set of classes by uniting both sets of concepts:

$$Con_1 := Con_{1,I} \cup Con_{1,E}$$

Now we have to build a matrix M, which represents the irreflexive binary relation $<$ defined on the generated class extensions expressing the specialization relation (subset relation) by comparing each pair of classes. A value '1' in M on row i and column j of M means that class C_i is a subclass of class C_j. If no subset relation between class C_i and class C_j exists then we write the value '0' into the corresponding matrix field. Complexity to build this matrix is again $O(n^3)$. Computing the matrix needs comparisons between at most $2 * n$ classes with at most $2 * n$ classes. For each comparison at most n base extensions have to be examined.

The computation $M_1 = M - M \times M$ removes transitive specializations and can be directly used to produce the integrated schema. A value '1' represents a non-transitive sub/super-class relation. Complexity to multiply matrices is $O(n^3)$.

Proposition. The proposed algorithm computes from a given context (G, M, I) representing the relation between base extensions and attributes the set of classes Con_1 with complexity $O(n^3)$. The set of classes Con_1 equals the set of valid concepts of the corresponding galois lattice. □

Proof. Let us assume, that the context (G, M, I) with $\mid G \mid = m$ and $\mid M \mid = n$ is given.

We proof this proposition in examining the two implications between both sets. The proof refers only to valid concepts with intensions of their own and to $Con_{1,E}$ because any proposition attributed to M holds for G, too.

1. *Each valid concept with attributes of its own corresponds to a class of* $Con_{1,E}$*:*

   A valid concept can encompass one or many attributes. If a concept has exactly one attribute (corresponding to a single row) then the corresponding class will be found by computing Ext_1 and $Con_{1,E}$.

   Suppose a valid concept c has the the extension $\{g_1^c, \ldots, g_a^c\}$ and the intension $\{m_1^c, \ldots, m_b^c\}$. If a concept encompasses more than one attribute, then two different cases are possible:

   (a) $\exists m_i^c \in [m_1^c, \ldots, m_b^c] : \forall g \in G : (g, m_i^c) \in I \Longrightarrow g \in [g_1^c, \ldots, g_a^c]$
   There is at least one attribute of class c to which *only* the base extensions of class c are assigned. Class c is found by computing $(\{m_i^c\}', \{m_i^c\}'')$ within Ext_1 and $Con_{1,E}$.

   (b) $\forall m_i^c \in [m_1^c, \ldots, m_b^c] : \exists g \in G : g \notin [g_1^c, \ldots, g_a^c] \wedge (g, m_i^c) \in I$
   To any attribute of class c at least one base extension is assigned which does not belong to the base extensions of class c. Each row m_i^c results in a concept $(\{m_i^c\}', \{m_i^c\}'')$ within Ext_1 and $Con_{1,E}$. The concept resulting from each attribute m_i^c has more objects (base extensions) than concept c and is, therefore, a superconcept (superclass) of concept c with the attribute m_i of its own. The attributes $\{m_1^c, \ldots, m_b^c\}$ of concept c are inherited from its superconcepts. Hence, concept c has no attributes of its own. This case cannot expose for a valid concept with attributes of its own.

2. *Each class of* $Con_{1,E}$ *corresponds to a concept with attributes of its own:*

   Suppose the class c of $Con_{1,E}$ is derived from the single attribute m_i. Class c has this attribute of its own if it is not inherited from one of its superclasses. This is always true because each superclass must have more base extensions than class c. Due to the computation of $\{m\}''$ no superclass can encompass attribute m_i.

From both implication follows the proposition. □

Chapter 5

Application to the Example

In this section we demonstrate the application of the proposed algorithm to our introduced example basing on Table 2.3. From each base extension g and from each attribute m we compute $\{g\}'$ and $\{m\}'$ and obtain Int_1 and Ext_1:

$$\begin{aligned}
Int_1 &= \{\{1\}', \{2\}', \{3\}', \{4\}', \{5\}', \{6\}', \{7\}', \{8\}', \{9\}'\} \\
&= \{\{Title, Author, ISBN\}, \\
&\quad \{Title, Author, Organ.\}, \\
&\quad \{Title, Author, ISBN, Publisher, Pubnr\}, \\
&\quad \{Title, Author, Organ., Volume\}, \\
&\quad \{Title, Author, Organ., Volume, TRnr, Pubnr\}, \\
&\quad \{Title, Author, TRnr, Pubnr\}, \\
&\quad \{Title, Author, Pubnr\}\} \\
Ext_1 &= \{\{Title\}', \{Author\}', \{ISBN\}', \{Organ.\}', \{Volume\}', \{Publisher\}', \\
&\quad \{TRnr\}', \{Pubnr\}'\} \\
&= \{\{1, 2, 3, 4, 5, 6, 7, 8, 9\}, \\
&\quad \{1, 3, 5\}, \\
&\quad \{2, 4, 6, 7\}, \\
&\quad \{4, 6, 7\}, \\
&\quad \{3, 5\}, \\
&\quad \{7, 8\}, \\
&\quad \{3, 5, 7, 8, 9\}\}
\end{aligned}$$

The sets $Con_{1,I}$ and $Con_{1,E}$ are derived from Int_1 and Ext_1:

$$\begin{aligned}
Con_{1,I} = \{&(\{1, 3, 5\}, \{Title, Author, ISBN\}), \\
&(\{2, 4, 6, 7\}, \{Title, Author, Organ.\}), \\
&(\{3, 5\}, \{Title, Author, ISBN, Publisher, Pubnr\}), \\
&(\{4, 6, 7\}, \{Title, Author, Organ., Volume\}), \\
&(\{7\}, \{Title, Author, Organ., Volume, TRnr, Pubnr\}), \\
&(\{7, 8\}, \{Title, Author, TRnr, Pubnr\}), \\
&(\{3, 5, 7, 8, 9\}, \{Title, Author, Pubnr\})\}
\end{aligned}$$

$$
\begin{aligned}
Con_{1,E} = \{&(\{1,2,3,4,5,6,7,8,9\}, \{Title, Author\}),\\
&(\{1,3,5\}, \{Title, Author, ISBN\}),\\
&(\{2,4,6,7\}, \{Title, Author, Organ.\}),\\
&(\{4,6,7\}, \{Title, Author, Organ., Volume\}),\\
&(\{3,5\}, \{Title, Author, ISBN, Publisher, Pubnr\}),\\
&(\{7,8\}, \{Title, Author, TRnr, Pubnr\}),\\
&(\{3,5,7,8,9\}, \{Title, Author, Pubnr\})\}
\end{aligned}
$$

After uniting both $Con_{1,I}$ and $Con_{1,E}$ we obtain the following classes:

$$
\begin{aligned}
\mathrm{C1} &= (\{1,2,3,4,5,6,7,8,9\}, \{Title, Author\})\\
\mathrm{C2} &= (\{1,3,5\}, \{Title, Author, ISBN\})\\
\mathrm{C3} &= (\{2,4,6,7\}, \{Title, Author, Organ.\})\\
\mathrm{C4} &= (\{4,6,7\}, \{Title, Author, Organ., Volume\})\\
\mathrm{C5} &= (\{3,5\}, \{Title, Author, ISBN, Publisher, Pubnr\})\\
\mathrm{C6} &= (\{7,8\}, \{Title, Author, TRnr, Pubnr\})\\
\mathrm{C7} &= (\{3,5,7,8,9\}, \{Title, Author, Pubnr\})\\
\mathrm{C8} &= (\{7\}, \{Title, Author, Organ., Volume, TRnr, Pubnr\})
\end{aligned}
$$

Now we have to build the matrix M (Tab. 5.1), which represents the irreflexive binary relation $<$ defined on the generated class extensions expressing the specialization relation (subset relation) by comparing each pair of classes.

$<$	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**
C1								
C2	√							
C3	√							
C4	√		√					
C5	√	√					√	
C6	√						√	
C7	√							
C8	√		√	√		√	√	

Table 5.1: Specialization relation matrix M

The computation $M_1 = M - M \times M$ (cf. Tab. 5.2) removes transitive specializations. A tick in M has to be replaced by '1' and no tick by '0'. In M_1 '1' represents by a tick and a value less than one no tick.

The set Con_1 and Table 5.2 can be directly used to produce the integrated schema (Fig. 5.1). The algorithm computes as an additional output the relation between new classes and the original classes, because we know about their sets of base extensions and attributes. This information is necessary, e.g., for query processing and giving the

<	C1	C2	C3	C4	C5	C6	C7	C8
C1								
C2	√							
C3	√							
C4			√					
C5		√					√	
C6							√	
C7	√							
C8				√		√		

Table 5.2: Specialization relation without transitive specializations: matrix M_1

derived classes suitable names. Most names of the derived classes can be taken over from the classes of the two existing databases and the database names.

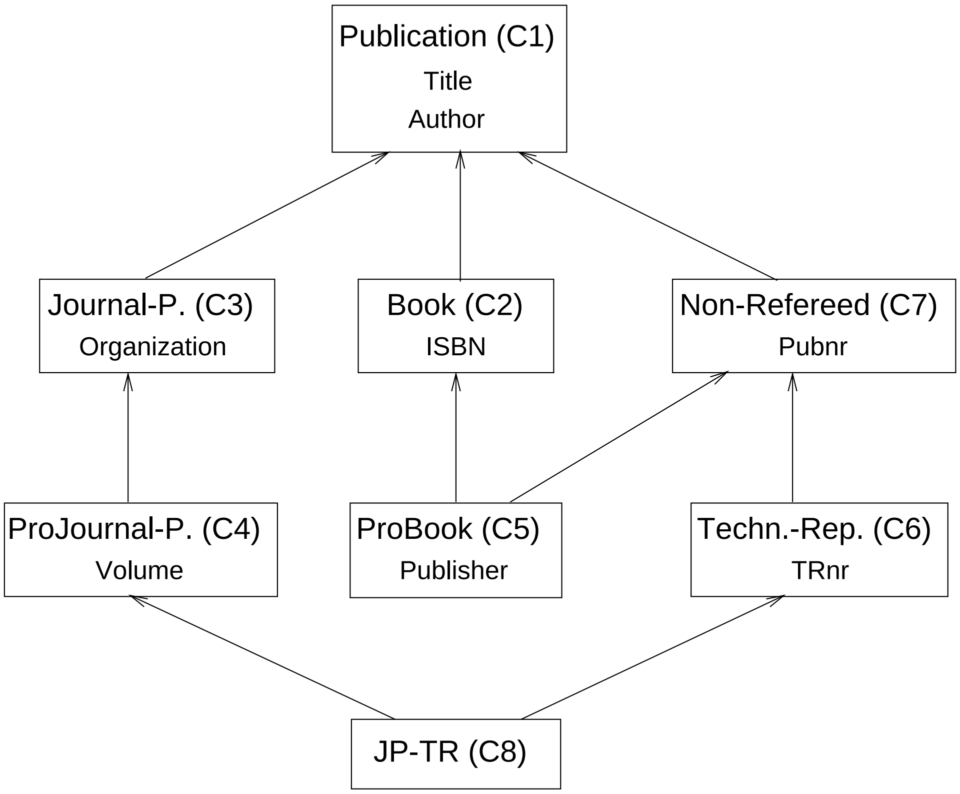

Figure 5.1: Integrated schema

Chapter 6

Transformation of Inheritance Hierarchies

The integrated schema resulting from the algorithm in Section 4 is the starting point of several optimization and transformation steps leading to the final integrated schema. These steps optimize the integrated schema with respect to different quality criteria (number of classes, number of specializations, removal of multiple specializations).

A number of different transformations is possible. Possible transformation steps are the following:

1. Dropping classes which do not have extensions but intensions of their own.
2. Dropping classes which do not have intensions but extensions of their own.
3. Vertically merging introducing null values.
4. Horizontally merging introducing null values.
5. Splitting of classes introducing artificial attributes.
6. Removing multiple specializations.

The items 1 to 4 are explained in this section. The remaining ones are discussed in Section 7.

A generated lattice may contain classes which do not have extensions of their own. In other words, for these classes their extensions are given by the union of their subclass extensions. Such classes can be dropped without loss of information. However, sometimes such a class improves the understandability and querying. In our example, the class `Publication` (C1) is a class which have an empty extension of its own.

There may exist other classes which can be dropped without loss of information. These classes are classes without attributes of their own and are generated by the proposed algorithm. However, if an object-oriented data model does not support the role concept (one object in more than one most specialized class; see [Bee93]) we cannot give up such classes. In our example such an class is the class `JP-TR` (C8).

An integration of inheritance hierarchies can produce a complex inheritance hierarchy. This complexity can be reduced if we admit null values. Admitting null values enables the merging of classes vertically and horizontally.

Vertically merging means merging a class with its direct superclass. The rectangle of the merged class is formed by the union of the extensions and of the intensions. The rectangle must be filled up by null elements which are considered as ticks. Then the algorithm has to be applied again. In our example we want to merge the classes `Journal-P.` (C3) and `ProJournal-P.` (C4). The united rectangle has extension (2,4,6,7) and the attributes (`Title`, `Author`, `Organization`, `Volume`). The adapted extension-attribute connection table is shown in Tab. 6.1. We introduced for extension **2** and attribute `Volume` the null element $\sim$.

Attr-Ext	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Title	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
Author	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
ISBN	$\surd$		$\surd$		$\surd$				
Organ.		$\surd$		$\surd$		$\surd$	$\surd$		
Volume		$\sim$		$\surd$		$\surd$	$\surd$		
Publisher			$\surd$		$\surd$				
TRnr							$\surd$	$\surd$	
Pubnr			$\surd$		$\surd$		$\surd$	$\surd$	$\surd$

Table 6.1: Extension-attribute relation with null element

The resulting schema of vertical merging is depicted in Fig. 6.1. The attribute `Volume` of the class `Journal-P.` is optional.

Analogously, classes can be merged horizontally. Classes can be merged horizontally if they have a direct specialization relation to the same superclass. The rectangle of the merged class is formed by the union of the subclass extensions and of the intensions. The rectangle must be filled up by null elements which are considered as ticks. Then the algorithm has to be applied again.

Figure 6.2 shows the result of horizontally merging the classes `Book` and `Non-Refereed`. We know from the extension-attribute relation of the merged class that each of its objects has a value for the attribute `ISBN` or `Pubnr` or both. With other words, this horizontally merging introduces an additional non-trivial integrity constraint.

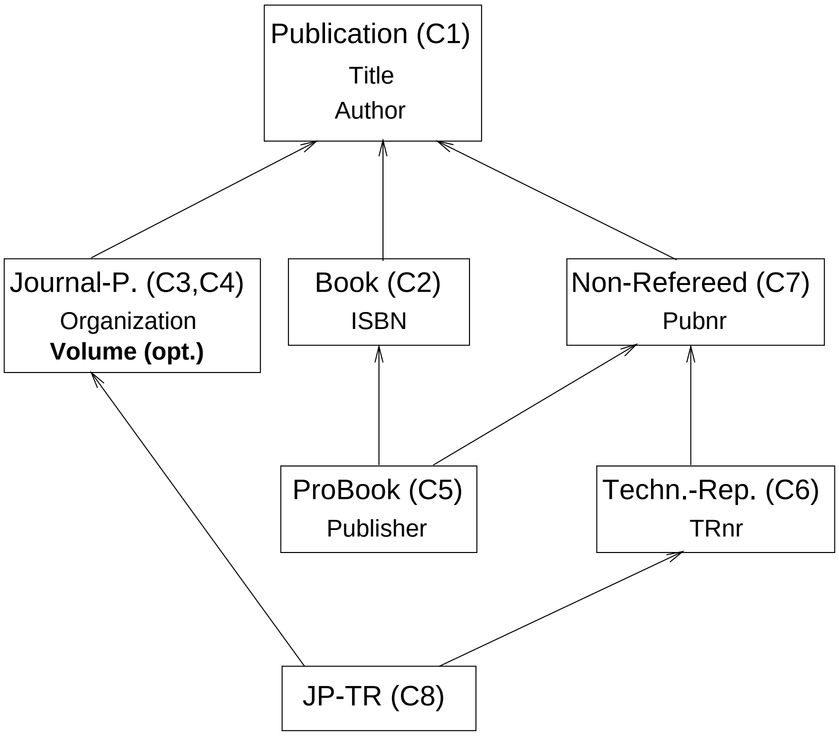

Figure 6.1: Result of vertically merging

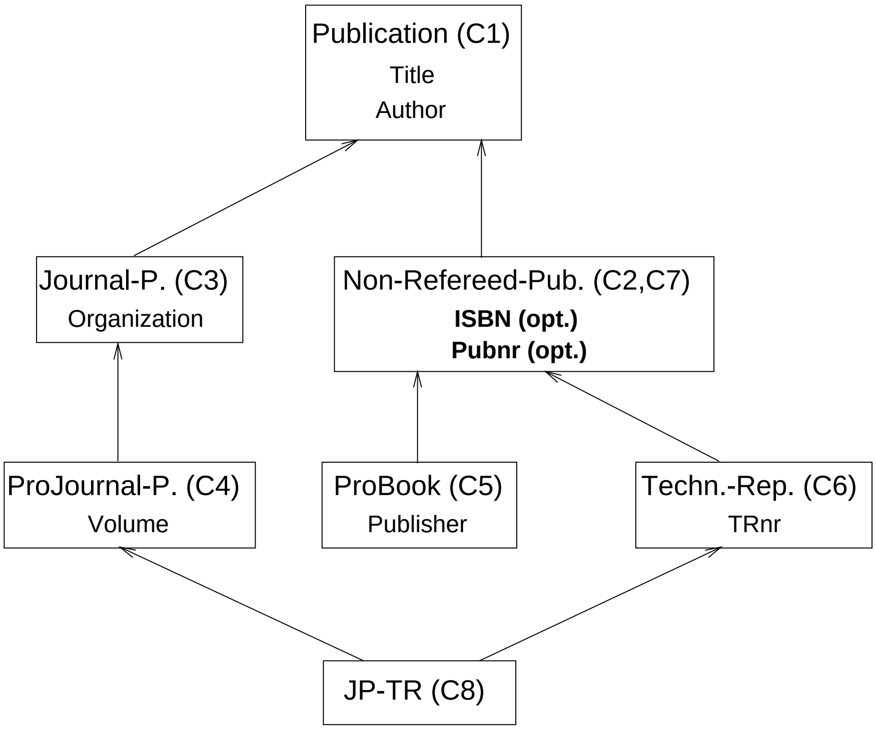

Figure 6.2: Result of horizontally merging

Chapter 7

Splitting of Inheritance Hierarchies

In the previous section we showed different ways to make an integrated schema as simple as possible, e.g. by minimizing the number of classes. However, these ways cannot be applied to all scenarios. In the area of schema integration specific conflicts must be considered. Sometimes, merging different class extensions of the input schema into one extension of a single integrated schema is not desired due to incompatible integrity conflicts. In such cases the designers have to influence the integration process to hinder a class merging. In these cases we propose to introduce artificial attributes in order to make the input classes more different.

In our example the extensions of the classes `Book` of both input schemata are merged into the integrated class `Book`. This can be avoided by introducing two artificial attributes `A1`, `A2` and thereby expanding the intensional overlappings (cf. Table 7.1). After applying the algorithm again we obtain an integrated schema (cf. Fig. 7.1) where the artificial attributes are omitted.

Attribute	**A1**	**A2**	**...**
Lib.Publication			...
Lib.Book	$\surd$		...
Lib.Journal-P.			...
Proj.Publication			...
Proj.Journal-P.			...
Proj.Non-Refereed			...
Proj.Book		$\surd$	...
Proj.Techn.-Rep.			...

Table 7.1: Additional intensional overlappings

The generated classes have the following extensions and intensions:

`Book` (C2) $= (\{1, 3, 5\}, \{Title, Author, ISBN\})$
`LibBook` $= (\{1, 3\}, \{Title, Author, ISBN\})$
`ProBook` (C5) $= (\{3, 5\}, \{Title, Author, ISBN, Publisher\})$

$\texttt{LibProBook} = (\{3\}, \{Title, Author, ISBN, Publisher\})$

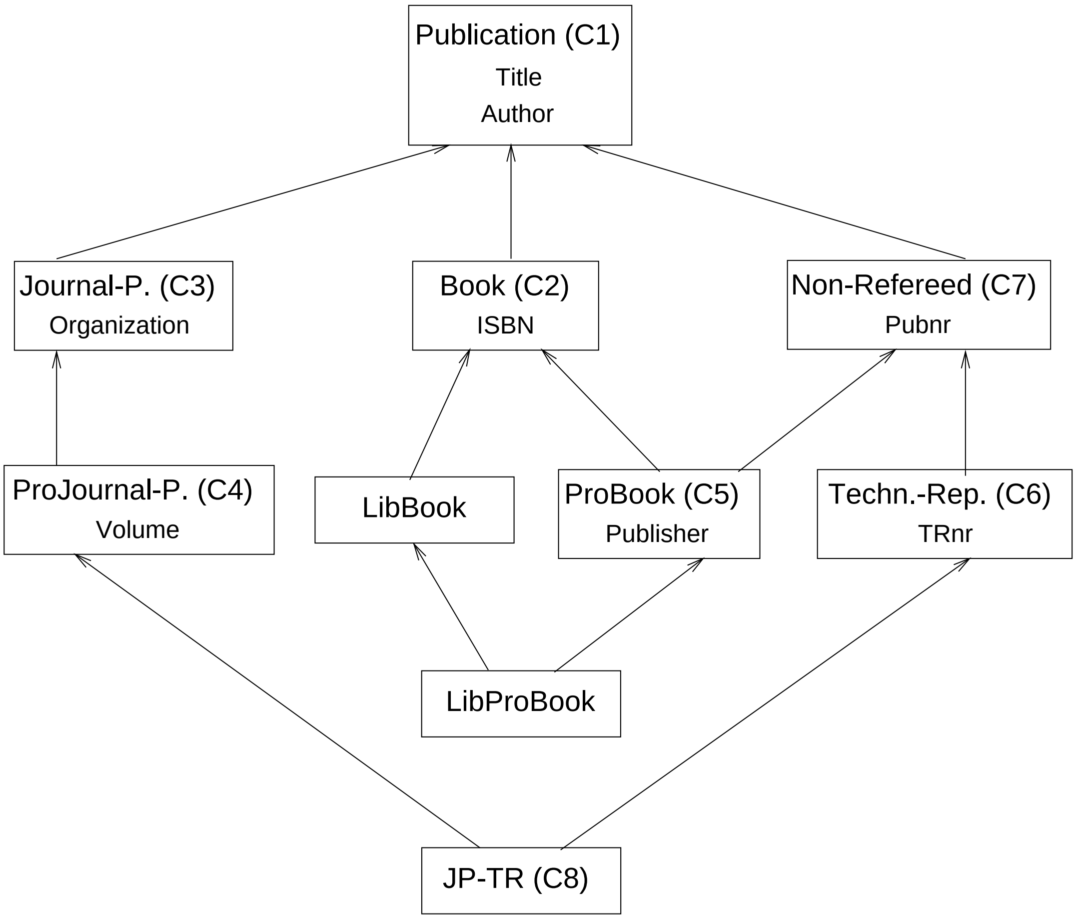

Figure 7.1: Splitting using artificial attributes

The presented algorithm generates inheritance hierarchies where multiple specializations (one class has more than one direct superclass) may occur. In further steps the integrated schema can be transformed into a schema without multiple specializations. On each class with more than one direct superclass the designer has to choose the direct superclass which should be the only superclass. Suppose class C1 is a direct subclass of the classes C2 and C3 and the designer want the class C1 to be subclass of C2 only. Then we have to modify the extension-attribute relation in the following way:

- $Ext(C2)$ is the set of base extensions of class C2
- $Ins(C2)$ is the set of attributes of class C2 (including inherited attributes)

Each row of the extension-attribute matrix which does not represent an attribute of $Ins(C2)$ must be duplicated to two row versions. The second version of the row has only ticks on each base extension (column) which belongs to $Ext(C2)$ and has a tick in the original row on the same base extension (column). Then, from the first (original) version of the row those ticks must be removed, which belong to $Ext(C2)$. After applying the algorithm to the changed extension-attribute matrix again we obtain a hierarchy, where the class C1 is the only subclass of C2. In the hierarchy we treat different versions of an attribute uniformly. In this way, all classes, which have more than one superclass can be sequentially transformed into classes of only one superclass.

Assume the designer of our example decided the class `ProBook` (C5) depicted in Fig. 5.1 has to be subclass of the class `Book` (C2) only.

$$Ext(C5) = \{3, 5\}$$
$$Ins(C5) = \{Title, Author, ISBN\}$$

In the extension-attribute matrix we have to duplicate the row `Pubnr` only because other duplications would produce empty row versions. By this way we obtain the matrix shown in Table 7.2.

Attr-Ext	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Title	√	√	√	√	√	√	√	√	√
Author	√	√	√	√	√	√	√	√	√
ISBN	√		√		√				
Organ.		√		√		√	√		
Volume				√		√	√		
Publisher			√		√				
TRnr							√	√	
Pubnr$_1$							√	√	√
Pubnr$_2$			√		√				

Table 7.2: Matrix with duplicated `Pubnr`

Our proposed algorithm produces a new integrated schema as depicted in Fig. 7.2.

We demonstrated here that we are able to transform an integrated schema into an equivalent schema without multiple specializations. In [SS96b, SS96a], another algorithm was shown which directly computes such schemata.

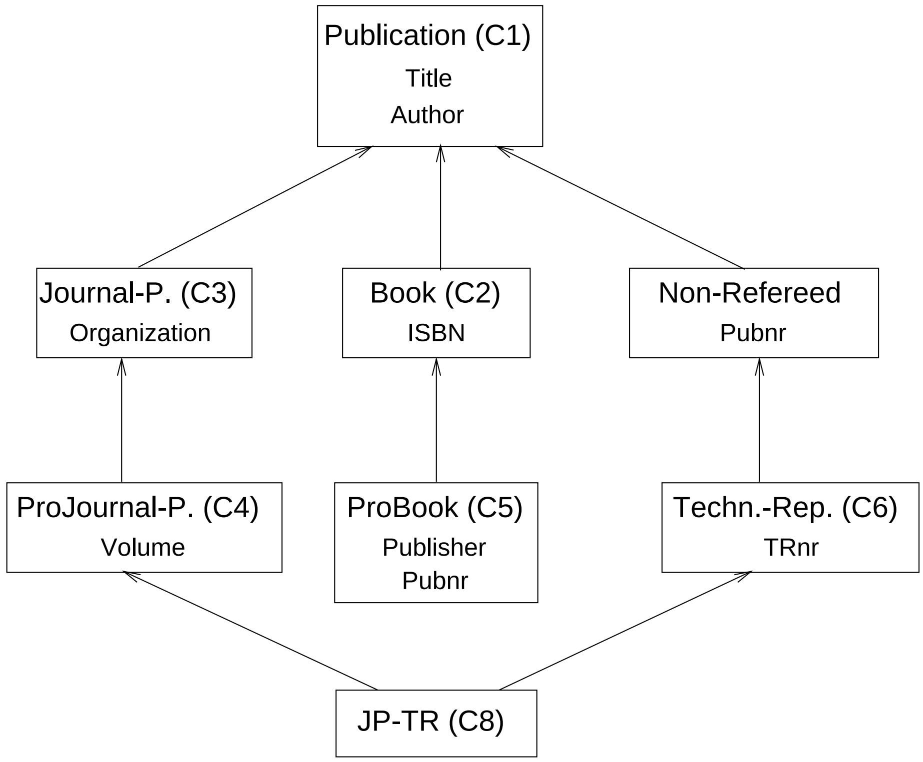

Figure 7.2: Result of removing a multiple specialization

Chapter 8

Conclusions and Outlook

The presented algorithms enable efficiently derivations of inheritance hierarchies from merged hierarchies adjusted to certain design requirements. Possible applications cover database integration, view integration, schema evolution and tuning of object-oriented schemata. The resulting schema can be influenced leading to schemata with specific properties while guaranteeing the correctness of the integrated schema.

For the problem of view and schema integration there exist various approaches, e.g. [GCS95, SP94, NG82]. In contrast to these works we provide a simple formal method for integration producing minimal (number of classes) integrated schemata. These advantages results from applying the theory of formal concept analysis (cf. [Wil92]) to problems of integration of views and schemata, respectively. This application requires a preintegration step of decomposing class extensions into base extensions (cf. [SS96a, SS96b]).

In [YLCB96] an algorithm is proposed supporting the design of an object-oriented schema. This algorithm is similar to our approach. The problem of empty class extensions and intensions was described, too. However, in [YLCB96] no extensional overlappings of input class extensions are considered. Furthermore, no formal comparison of the resulting inheritance hierarchy with a concept lattice is given.

Future work will include a modification of the algorithm for deriving specific views which present parts of the database. The modified algorithm allows to complete a partial view description towards a complete hierarchy. The input information for the algorithms, especially the information about extensional overlappings, can be partially derived using data inspection of existing databases. Other relevant information can be derived from schema information and integrity constraints. We plan to develop methods for extracting such information for different data models as part of a project on database federation [SCH+96, SEHT96, SCC+97].

Bibliography

[Bee93] C. Beeri. Some Thoughts on the Future Evolution for Object-Oriented Databases Concepts. In W. Stucky and A. Oberweis, editors, *Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'93), Braunschweig, März 1993*, pages 18–32. Informatik aktuell, Springer-Verlag, Berlin, 1993.

[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.

[CTK96] A. L. P. Chen, P. S. M. Tsai, and J.-L. Koh. Identifying Object Isomerism in Multidatabase Systems. *Distributed and Parallel Databases*, 4(2):143–168, April 1996.

[Dup94] 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.

[Duq87] V. Duquenne. Contextual implications between attributes and some properties of finite lattices. In B. Ganter and R. Wille, editors, *Beiträge zur Begriffsanalyse*, chapter 10, pages 213–239. B. I.-Wissenschaftsverlag, Mannheim, 1987.

[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.

[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, 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.

[NEL86] 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.

[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.

[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.

[SCC+97] G. Saake, A. Christiansen, S. Conrad, M. Höding, I. Schmitt, and C. Türker. Föderierung heterogener Datenbanksysteme und lokaler Datenhaltungskomponenten zur systemübergreifenden Integritätssicherung — Kurzvorstellung des Projekts $\mathbf{SIGMA}_{FDB}$. In *Proc. GI-Fachtagung "Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'97), Ulm, März 1997*. Informatik aktuell, Springer-Verlag, Berlin, 1997.

[SCH+96] G. Saake, S. Conrad, M. Höding, S. Janssen, I. Schmitt, and C. Türker. Föderierung heterogener Datenbanksysteme und lokaler Datenhaltungskomponenten zur systemübergreifenden Integritätssicherung — Grundlagen und Ziele des Projektes $\mathbf{SIGMA}_{FDB}$. In *Integrationskonzepte für Mensch, Technik, Organisation, und Planung — Grundlagen und Projektziele der Projektgruppe METOP, Bericht zum 1. Kolloquium*, pages 13–24. Preprint Nr. 5, Fakultät für Maschinenbau, Universität Magdeburg, 1996.

[SEHT96] I. Schmitt, A. Ebert, M. Höding, and C. Türker. $\mathbf{SIGMA}_{Bench}$ – Ein Werkzeug zum Entwurf f"oderierter Datenbanken. In W. Hasselbring, editor, *Kurzfassungen zum 2. Workshops "Föderierte Datenbanken", Dortmund, 12.-13. Dezember 1996*, pages 19–26. SWT Memo Nr. 90, Fachbereich Informatik, Universität Dortmund, 1996.

[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 P. 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.

[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*, pages 195–210. LNCS 1157, 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*, pages 751–758, September 1996.

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

[Wil92] R. Wille. Concept lattices and conceptual knowledge systems. *Computer & Mathematics with Applications*, 23(6-9):493–515, 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*, pages 422–437. LNCS 1157, Springer-Verlag, Berlin, 1996.