Acta Informatica (2005) 41(7/8): 475–524
DOI 10.1007/s00236-005-0166-2

ORIGINAL ARTICLE

**Ingo Schmitt · Gunter Saake**

# A comprehensive database schema integration method based on the theory of formal concepts

Received: 10 December 2001 / Revised: 22 February 2005 / Published online: 24 May 2005
© Springer-Verlag 2005

**Abstract** Integrating heterogeneous database schemata is a major task in federated database design where preexisting and heterogeneous database systems need to be integrated virtually by providing a homogenization database interface. Most proposed schema integration methods suffer from very complex result schemata and insufficient handling of extensional relations, i.e. in the way how redundant data of the input systems are dealt with. Redundancy among the input systems may thus remain undetected and, hence, remains uncontrolled.

Our GIM (Generic Integration Model) method is based on the elegant and mathematically founded theory of *formal concept analysis (FCA)*. The main idea is to integrate schemata into one *formal context* which is a binary relation between a set of attributes and a set of base extensions (set of potential objects). From that context we apply an FCA-algorithm to semi-automatically derive a concept lattice which we interpret as an inheritance hierarchy of classes for a homogenized schema. Thus, the integration task following our method can be supported by tools.

## 1 Introduction

Assume two database systems are given and a global application needs data from both databases. The database systems are usually heterogeneous w.r.t. to the database management system, the database model, and the database schema requiring an elaborate homogenization for global accesses. Instead of

I. Schmitt (✉) · G. Saake
Otto-von-Guericke-Universität Magdeburg, Institut für Technische und Betriebliche Informationssysteme, PF 4120, 39016 Magdeburg, Germany
E-mail: {schmitt, saake}@iti.cs.uni-magdeburg.de

performing an individual homogenization for every global application, it is often better to integrate the database systems only once, physically or virtually. In this case, we need a schema integration method which homogenizes and integrates two database schemata into one database schema.

The starting point of a schema integration process are therefore two preexisting, possibly heterogeneous database systems with their schemata. Additionally, the designer has knowledge about the semantics of the underlying databases, that is, he has an understanding of the semantical links of schema elements to concepts of the real world.

The main goal of the integration process is to construct an *integrated schema* for the databases to be integrated. Similar to database views in traditional database systems, the derivation of several *external schemata* and their adaption to application-specific needs must be supported. Schema integration works as a template for the integration of the underlying databases. Therefore, we need processable information about the mappings of the external schemata to the preexisting schemata as result from the schema integration process. Figure 1 depicts the integration process as black box with its inputs and outputs.

Database schema integration has been a major topic of database engineering since the early days of database management systems. Problems of database schema integration occur in several scenarios, e.g. during view integration within database design, integration of databases in data warehouses, designing federated databases, and database schema extension during schema evolution.

In this paper, we concentrate on database schema integration as part of constructing a federated database [73]. A federated database system (FDBS) establishes a virtual global database interface on preexisting database systems while keeping the participating systems autonomous. Problems of federated database systems have been discussed since the eighties of the last century and are relevant in many database projects where databases need to be integrated, e.g. also in data warehousing projects.

From the view of database schema integration, designing an FDBS is a complex challenge. In addition to schema information we have to deal with database states manipulated autonomously by local applications. We have to solve intrinsic data heterogeneity of databases populated by different companies and organizations. Since the integration is virtual, we have to define database schema mappings processable by an FDBMS. Last but not least, some scenarios even require

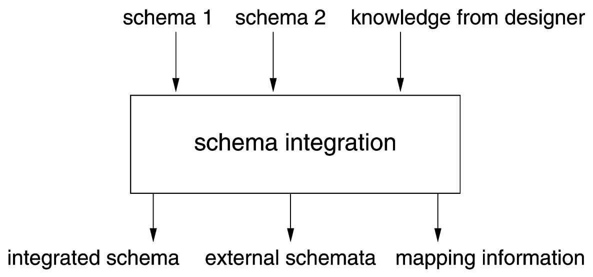


**Fig. 1** Input and output of the schema integration.

global updates which require additionally an inverse mapping of database states and updates.

Besides the problem of homogenization, we have to face the problem of *redundancy* arising from independently developed and populated databases. Parts of information may be stored redundantly in several databases. Such kind of redundancy has to be considered for a correct integration.

We present the *Generic Integration Model (GIM) method* as a new approach to schema integration. Following our approach, in contrast to many other approaches, preexisting database schemata are *not directly* mapped onto one homogenized result schema. Instead, they are together mapped firstly into *one* intermediate representation being an instance of the *Generic Integration Model* (GIM). From this representation, which is regarded as a *formal context* [23], we apply secondly an algorithm to semi-automatically derive the final integrated database schema. With respect to the notion of information capacity, see for example in [32], there is no unwanted data loss caused by mapping the preexisting database schemata onto a GIM representation. However, information of *how* the data behind the input schemata *are presented* gets lost. In order to reconstruct an appropriate overall presentation we apply an algorithm producing a *concept lattice*. We interpret every concept as a class of the homogenized result schema and the underlying partial order of the lattice as specialization relationship between classes. The algorithm is optimized to find a minimal number of classes. The designer can manipulate the derivation process to tailor the result schema to specific design goals. Furthermore, besides schema mappings, our GIM method produces corresponding data mappings.

The GIM integration method covers the whole integration process required for an FDBS design. Main steps are formalized and can therefore be supported by design tools which themselves can produce processable transformation rules for data conversion, queries and updates. For comparison with other integration methods, we exemplarily relate our schema transformations to SIG-transformations as published in [47,48].

For the following presentation, we restrict our consideration to the integration of *two* preexisting databases. For its generalization to an n-ary case we refer to [2]. We concentrate furthermore on structural (i.e. static) aspects and will not discuss dynamic aspects of database integration such as behavior integration [53]. Due to space limitations we focus on explaining and defining key steps of the GIM approach. For a more detailed discussion we refer to [68].

Before starting the technical part of our approach, we provide some definitions of terms frequently used in subsequent sections:

- Redundancy between two databases is expressed by establishing a binary SAME *relation* on two sets of database objects. The SAME relation is an equivalence relation. For example, data about the same person can be stored in different databases. The corresponding database entries are interrelated by the SAME relation. Figure 2 depicts the case where two different database entries model the same real world entity.
- Database objects are grouped into classes. We will use the term *intension* interchangeably for the type of a class, that is, for the set of its typed attributes.
- The *extension* of a class $c$, denoted as $\mathcal{E}xt_c$, is defined as the set of instances (objects) of $c$ at a certain instant of time.

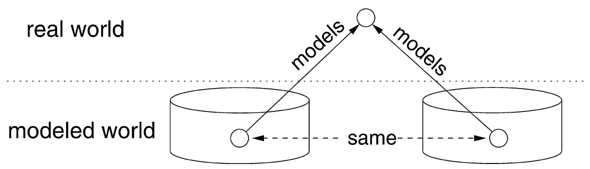


**Fig. 2** SAME relation.

- *Extensional analysis* is the non-trivial process of inspecting class extensions, application programs and developer knowledge to detect extensional relationships between different classes without explicitly knowing all possible database states in future. An example of an extensional relationship is the information that for every possible object of a database class there exists exactly one object of another database class which is semantically related by the SAME relation. Such extensional relationships must be valid independently from specific database states, i.e. from certain instants of time. They may be specified by extensional *assertions* like inclusion, disjointness, and overlap of the corresponding class extensions. An inclusion assertion, for example, may state, that every person stored in one database is always simultaneously stored in another one.
  The classical approach to extensional analysis is to state extensional assertions binarily only, for example stating that the classes `Man` and `Woman` are always disjoint. Later on, we will discuss the deficiencies of being restricted to *binary* extensional assertions.

This paper is organized as follows. Section 2 discusses the overall requirements for schema integration and the quality criteria for the result schemata and their mappings. Section 3 presents an example to be used in the remaining sections. Additionally, the specific problems arising in the example scenario are discussed. Section 4 describes the main ideas and concepts, including the generic integration model, behind our integration method. The translation of the input schemata into GIM is described in Sect. 5. The GIM representations need to be compared in order to find conflicting schema elements. Section 6 shows how to resolve such conflicts by transforming schemata. After this step the resulting two GIM representations are merged into one representation in Sect. 7. To obtain an integrated view in a database model, Sect. 8 discusses ways of deriving external schemata from the integrated GIM representation. An external schema can be adapted to a certain application view. After discussing related work in Sect. 9 we conclude and summarize our work in Sect. 10.

## 2 Requirements for schema integration and schema levels

Our integration scenario implies some intrinsic problems that have to be resolved by an appropriate integration method:

1. The two input database schemata may be formulated in different database models. This database model heterogeneity is resolved by translating them into a *common data representation* during the integration process.
2. Parts of the translated schemata may be semantically interrelated. That is, they can refer to same real world concepts and are in this sense redundant, e.g. a

class `Employee` can occur in different schemata. On data level, such redundant schema parts often cause their underlying database objects being semantically related by the SAME relation.
3. The problem of *schema heterogeneity* arises because redundant schema parts may be modelled differently.
4. The designer knowledge about the semantics of the preexisting databases is often neither explicitly available, exact, consistent, nor complete. Finding redundant schema parts is therefore not a trivial task.

These problems lead to some (partly formalized) requirements for the result of the integration process. They were published in [2] and include:

1. *completeness and correctness:* The integrated schema must contain all concepts present in any input schema correctly. The integrated schema must be a representation of the union of the application domains associated with the schemata.
2. *minimality:* Although the same concept may be represented in more than one input schema, it must be represented only once in the integrated schema.
3. *understandability:* The integrated schema should be easy to understand for the designer and the end user. This implies that among the several possible representations of results of integration allowed by a data model, the one that is (qualitatively) the most understandable should be chosen.

These requirements will be discussed in the following sections in more detail. Before doing this, we give some details of the integration process and identify some intermediate schema representations.

## 2.1 Schema levels

We introduce different schema levels for the integration process and start with the two input schemata $S_1$ and $S_2$ at the lowest level. Final result of the integration process are one integrated representation $S^I$ and possibly several external schemata $S^E$ as schemata at the highest level. We propose to use additional schema representations at intermediate levels:

1. The two input schemata $S_1$ and $S_2$ are translated into a common data representation used for the integration. This step overcomes data model heterogeneity and produces the component schemata $S_1^C$ and $S_2^C$.
2. In order to overcome schema heterogeneity the schemata $S_1^C$ and $S_2^C$ are *homogenized* resulting in the two homogenized schemata $S_1^H$ and $S_2^H$.
3. The homogenized schemata $S_1^H$ and $S_2^H$ are *merged* into *one* integrated schema $S^I$.
4. Several external schemata $S_1^E, \ldots S_i^E$ are derived from $S^I$ in order to reflect different application-specific needs and restrictions.

The five-level schema architecture from Sheth and Larson [73] is usually accepted as a reference model for a database federation. Figure 3 relates our schema levels (right) to those of [73] (left).

As a difference, we do not utilize export schemata but homogenized schemata $S^H$. Furthermore, our first merged schema $S^I$ is expressed in the Generic Integration Model und, thus, is no database schema for an end-user.

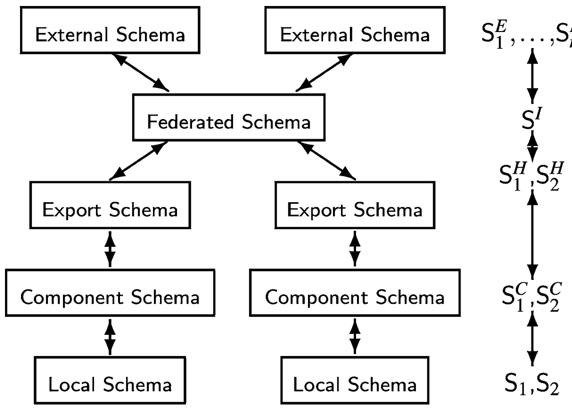


**Fig. 3** Schema architecture.

**Table 1** Requirements for schemata at different levels

| Schema | Understandability | Minimality |
|---|---|---|
| $S^E$ | required | required |
| $S^I$ | not required | required |
| $S_1^H, S_2^H$ | not required | not required |
| $S_1^C, S_2^C$ | not required | not required |

Table 1 relates the requirements for understandability and minimality to our different schema levels. These requirements are subjective and can, therefore, hardly be formalized.

In order to characterize completeness and correctness more precisely we need to consider mappings between schema instances (database states). Let $S$ be a schema, $I(S)$ the set of all possible instances of a schema $S$, and $Sym(i)$ the set of all symbols, i.e. all attribute values of an instance $i \in I(S)$. Furthermore, we define for $j \in \{1,2\}$ the database mappings $\varphi_j^C$ from $I(S_j)$ to $I(S_j^C)$, $\varphi_j^H$ from $I(S_j^C)$ to $I(S_j^H)$, $\varphi^I$ from $(I(S_1^H) \times I(S_2^H))$ to $I(S^I)$, and $\varphi^E$ from $I(S^I)$ to $I(S^E)$.

In the database theory, there is the concept of *schema equivalence* , c.f. [32, 48]: A database mapping $\varphi$ between instances of $S_1$ and $S_2$ is said to be $Z$*-internal* for a given value set $Z \subseteq DOM$, if $\varphi$ is functional, injective, total, and $\forall i \in I(S_1) : Sym(\varphi(i)) \subseteq Sym(i) \cup Z$ holds. The schemata $S_1$ and $S_2$ are defined to be $Z$*-internal equivalent*, if a finite set $Z \subseteq DOM$ and a mapping $\varphi$ between $I(S_1)$ and $I(S_2)$ exists which is $Z$-internal and bijective. This kind of equivalence implies that all new values introduced by a bijectively mapped database instance must be elements of the finite set $Z$.

For completeness and correctness we require $Z$-internal equivalence among $S_j$, $S_j^C$, and $S_j^H$. We require furthermore $Z$-internal equivalence except injectivity between the combined schemata $(S_1^H, S_2^H)$ and $S^I$ as well as between $S^I$ and $S^E$. Injectivity is dropped in order to overcome redundancy by merging instances from $S_1^H$ and $S_2^H$. For example, the address of the same employee may be stored simultaneously in two different databases. These address values should be equal but due to possibly different degrees of data quality they may be different. Merging the corresponding database states means to map the two address values onto one value. Since one value gets lost injectivity of the mapping cannot be fulfilled. Therefore, we require injectivity to be abandoned in order to overcome data conflicts among redundant data.

Since an external schema should reflect a user-specific *part* of the database injectivity is abandoned in this case, too.

### 2.2 Deficiencies of existing integration methods

Schema integration is one bottleneck of building database federations [50]. One reason can be found in common deficiencies of most existing integration methods[1]: (i) Most approaches consider only *some* steps of the integration process and do not present a method covering all steps from heterogeneous input databases up to the external schema derivation. (ii) In general, integrating heterogeneous schemata is a complex task. Unfortunately, most integration approaches do not provide design algorithms to support this task. (iii) Most approaches suffer from insufficient dealing with extensional conflicts. Extensional conflicts express potential redundancy among classes. That means, data about one real-world object can be stored in more than one class simultaneously. To meet the requirements for minimality and correctness we have to know how class extensions are semantically related to each other. Most approaches enable the specification of binary extensional assertions comparing the extensions of two classes only. As shown in [66] *binary* assertions are not powerful enough to express all possible extensional relations among classes. Therefore, some SAME objects may remain undetected or some global class extensions may be always empty, and thus superfluous.

Some approaches, e.g. [28,56,57], do not any analyze extensional relations. They merge semantically-related classes simply into one global class. The set of attributes of the global class equals the union of the local class attributes. Without extensional analysis the designer does not know for which potential global object there is a value for an attribute. Therefore, many null values occur. Managing null values usually requires a special treatment and should, therefore, be avoided.

Our approach aims to overcome these problems. It covers the whole integration process and provides: (i) a complete and correct handling of extensional analysis (presented in Sect. 6.2); (ii) an elegant algorithm to support the derivation of external schemata (Sect. 8). (iii) mechanisms to manipulate the derivation process in order to derive an external schema which is tailored to application-specific needs.

## 3 An integration example

For demonstration purposes we utilize a running example while explaining our integration method. Figure 4 depicts two database schemata in an UML-like notation. The first database stores information about people working in a company.

[1] A detailed discussion of related work is given in Sect. 9.

**Table 2** Integrity constraints

| Local Class | Integrity Constraint |
|---|---|
| $\texttt{Manager}_1$ | $\texttt{salary} \geq 3500$ |
| $\texttt{Employee}_2$ | $\texttt{salary} \geq 3000$ |

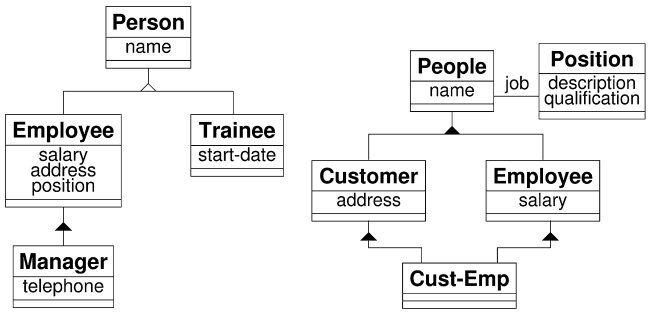


**Fig. 4** Schema 1: company schema and schema 2: department schema.

Every person is either an employee or a trainee. Furthermore, every manager is an employee. The second database contains data about persons relevant for a department of this company. Every person contained in `People` of the department database is a customer, or an employee, or both of them (contained in class `Cust-Emp`). We will distinguish classes from both databases using subscripts following their names.

Table 2 lists specified integrity constraints.

The example includes an attribute conflict w.r.t. the attribute `name`. Attribute $\texttt{name}_1$ contains last names whereas $\texttt{name}_2$ shall contain full names.

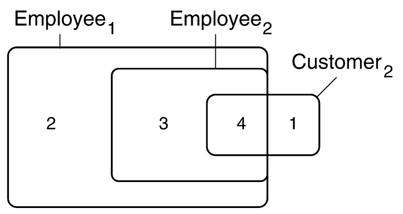
For our small example, we assume that the following extensional relations[2] hold: (1) Every employee from class $\texttt{Employee}_2$ is simultaneously contained in class $\texttt{Employee}_1$. (2) Every person entry simultaneously contained in $\texttt{Employee}_1$ and $\texttt{Customer}_2$ is also an instance of class $\texttt{Employee}_2$ (and thus contained in class $\texttt{Cust-Emp}_2$).

Besides for demonstration purposes, we will use this small example to discuss the problems if only *binary* extensional assertions are utilized for the extensional analysis. As a first observation, we realize that both assertions involve certain overlaps between the extensions of the classes $\texttt{Employee}_1$, $\texttt{Employee}_2$ and $\texttt{Customer}_2$. These overlaps are graphically depicted in Fig. 5 (using an Euler/Venn diagram notation).

On the right hand side of Fig. 5, we give an equivalent notation for expressing extensional overlaps which does not have the intrinsic restrictions of a diagrammatic notation. Such an *extension diagram* splits all extensions into disjoint *base extensions* (labelled with numbers) and specifies the relation between class extensions and base extensions as a Boolean matrix.

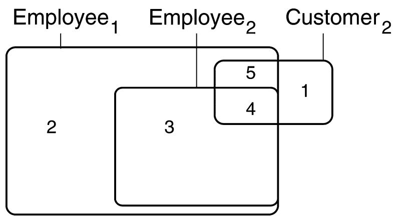
Obviously, the second assertion cannot be exactly stated by binary assertions. If we are forced to binary ones we can just state that $\texttt{Employee}_1$ can overlap

[2] In addition to those which are already stated by the given specialization among classes.



| Local Class | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Employee$_1$ | | ✓ | ✓ | ✓ |
| Employee$_2$ | | | ✓ | ✓ |
| Customer$_2$ | ✓ | | | ✓ |

**Fig. 5** Exact extensional relationships among Employee$_1$, Employee$_2$, and Customer$_2$.



| Local Class | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Employee$_1$ | | ✓ | ✓ | ✓ | ✓ |
| Employee$_2$ | | | ✓ | ✓ | |
| Customer$_2$ | ✓ | | | ✓ | ✓ |

**Fig. 6** Extensional relationships among Employee$_1$, Employee$_2$, and Customer$_2$ due to binary assertions.

Customer$_2$ and Employee$_2$ can overlap Customer$_2$. The result is depicted in Fig. 6. It introduces an additional base extension (5), which is always empty and, therefore, superfluous. That wrong base extension is a direct consequence of using exclusively binary assertions as proposed by many integration methods. Besides the GIM approach, only a few other approaches, e.g. [8,17], allow for extensional assertions of higher arity than binary.

## 4 The GIM method

This section presents the core ideas and concepts behind our integration method. As stated earlier, the main problem of database schema integration is the difficulty of resolving schema heterogeneity. Schema conflicts occur when redundant parts of various schemata are modelled differently. Such conflicts occur very often since a database model offers a database designer typically a certain freedom to model one real world concept in different ways. Obviously, the more modelling concepts are available the more probable do schema conflicts occur and, thus, the more difficult is the task to overcome them. Therefore, using a semantically rich database model with many modelling concepts requires often an expensive homogenization.

For expressing the integrated schema, one particular database model must be chosen to which the input schemata need to be translated. In order to avoid a loss of any schema information, most integration methods have chosen a semantically rich database model. Such a model provides enough modelling concepts to be the appropriate target database model for translating schemata of, hopefully, any possible database model.

Summarizing this discussion, we are confronted with two conflicting goals when we have to choose an appropriate database model as integration model. At the one hand, it should be as semantically rich as possible in order to avoid loss of schema information. At the other hand, many modelling concepts make the homogenization process very complex and, furthermore, result in very complex and unhandy integrated schemata.

Most integration methods ignore the deficiencies of choosing a semantically rich database model and choose, therefore, an object-oriented or object-relational database model as integration model.

Our integration method, however, respects both goals. The conflict is resolved by introducing a special data model for the intermediate schemata $S^C$, $S^H$, and $S^I$ (see Fig. 3 on p. 480). This database model is intended to serve as an integration model but *not* as an end user database model. We have chosen the *generic integration model* (GIM) as integration model. In order to make the homogenization as easy as possible, GIM offers only a relatively small set of modelling concepts. GIM is a class-based data model supporting disjoint extensions, i.e., GIM is not intended to support a specialization of classes.

That decision provokes the question about the potential loss of schema information when an input schema expressed in a semantically rich database model needs to be translated into GIM. In order to answer that question, we distinguish between two kinds of schema information loss:

1. *data loss*: Transforming a database state in accordance to a schema translation can produce a loss of data if the corresponding schemata are not Z-internal equivalent.[3]
2. *loss of presentation data*: Presentation data of a schema define the way *how* an underlying database is presented to the end user.

Our integration model GIM is designed to avoid loss of data, but presentation data can get lost. The rationale behind neglecting loss of presentation data is twofold: (1) If, in general, a schema conflict needs to be resolved then obviously at least one way of presentation must be abandoned anyway. Therefore, in principle, loss of presentation data cannot be avoided completely. (2) Our integration method provides an FCA-algorithm to semi-automatically construct new presentation data for an external schema on top of a integrated GIM schema.

Our integration method, therefore, performs in a first phase a data lossless schema translation, homogenization, and integration involving GIM. The second phase, however, comprises a semi-automatically construction of appropriate presentation information for an external schema from the integrated GIM schema. Our experiences with that construction of presentation information have shown that our algorithm produces relatively easy and understandable external schemata. Additionally, the designer can manipulate the derivation algorithm in order to tailor an external schema to application-specific needs. This includes even the case when the designer wants to recover lost presentation information as much as possible.

Besides our running example, the following very small example sketches our integration method. Figure 7 depicts two schemata to be integrated. The first one contains the class `Student` and the second one the classes `Person` and `Employee`. Due to an extensional analysis we may know that the extension of `Person` can be partitioned into a student and an employee extension.

[3] For the definition of Z-internal equivalence see Sect. 2.

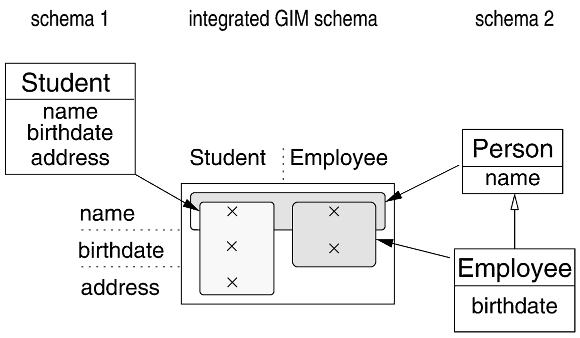


**Fig. 7** Mapping of two input schemata into one integrated GIM schema.

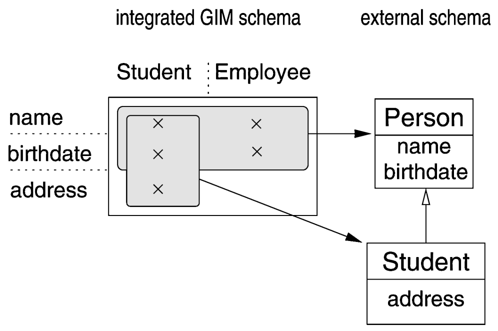


**Fig. 8** Deriving an external schema from an integrated GIM schema.

Furthermore, we assume[4] attributes with same names share same semantics. Using this information, the input schemata are mapped into one integrated GIM schema. There, a row represents an attribute and a column refers to a base extension. A cell at a certain row and a certain column is set if for that kind of objects a value for the corresponding attribute would exist. The classes of the input schemata can be represented by rectangles.

From the integrated GIM schema we are now able to derive an external end user schema by looking for maximal rectangles filled with crosses. In our small example, see Fig. 8, we obtain the classes `Person` and `Student`. Their overlapping determines the class `Student` to be a subclass of `Person`. The output class `Person` subsumes the input classes `Person` and `Employee`.

## 4.1 Formal definition of GIM

GIM is a class-based data model for modelling classes with *disjoint* extensions. Attributes have atomic data types. Integrity constraints like uniqueness, cardinality, and domain constraints can be specified. GIM supports bidirectional reference attributes, i.e. reference attributes occur pairwisely. In this way, during the derivation of an external schema one can freely choose a certain reference direction (as specific presentation detail) by dropping the inverse reference attribute.

[4] This assumption holds only for this little example.

We assume that the set $\mathcal{D}at$ and the set function $\mathcal{D}omain$ are given. $\mathcal{D}at$ contains names of data types and reference data types. The set-valued function $\mathcal{D}omain$ is a function over $\mathcal{D}at$. $\mathcal{D}omain$ assigns a set of values to each data type name. A value of a reference data type is defined to be a set of object identifiers.[5]

The constituents of GIM are defined as follows:

- Class $C \overset{\text{def}}{=} (Cname, Att, IC)$: $Cname$ is the name of the class and $Att$ is a function from a set of attribute names into $\mathcal{D}at$ defining the intension (type) of a class. Each class includes an attribute Id which represents the object identifiers. $Att$ can also contain reference attributes. Every reference attribute defines a set of referred class and must correspond to an inverse reference attribute. Additionally, $IC$ contains local uniqueness constraints, domain constraints and cardinality constraints.
- Schema $S \overset{\text{def}}{=} (Sname, \mathcal{C}, Unique)$: $Sname$ is the name of the schema and $\mathcal{C}$ is a set of classes $\{C_1, C_2, \ldots, C_n\}$. $Unique$ is a set of global uniqueness constraints ranging over several classes.
- Extension $\mathcal{E}xt_C$ of a class $C$: $\mathcal{E}xt_C$ is defined as a set of functions. Each function represents an object and defines the attribute values for the given attributes:

  $$\begin{aligned}&\forall f \in \mathcal{E}xt_C : f \text{ is a function over } dom(C.Att) \wedge \\ &\forall x \in dom(f) : f(x) \in (\mathcal{D}omain(C.Att(x)) \cup \{NULL\}).\end{aligned}$$

  The object identifiers have to be unique:

  $$\forall f_1, f_2 \in \mathcal{E}xt_C : f_1(Id) = f_2(Id) \Rightarrow f_1 = f_2.$$

- The extension $Ext_{Id,C}$ contains the identifiers of the extension of class $C$:

  $$Ext_{Id,C} \overset{\text{def}}{=} \{f(Id) \mid f \in \mathcal{E}xt_C\}$$

  $Ext^t_{Id,C}$ denotes the extension at time $t$.
- A database state $\mathcal{S}tate_S$ is a set-valued function over the classes $\mathcal{C}$ of schema $S$.[6] A state assigns to each class of the schema a class extension:

  $$\forall C \in S.\mathcal{C} : \mathcal{S}tate_S(C) = \mathcal{E}xt_C$$

  Additionally, we require extensional disjointness:

  $$\forall C_1, C_2 \in S.\mathcal{C} : C_1 \neq C_2 \Rightarrow Ext^t_{Id,C_1} \cap Ext^t_{Id,C_2} = \emptyset$$

- For a pair of inverse reference attributes $r_1$ and $r_2$ of classes $C_1$ and $C_2$, respectively, we require:

  $$\begin{aligned}&\forall o_1 \in \mathcal{S}tate_S(C_1) : \forall oid \in o_1(r_1) \Rightarrow \\ &\exists o_2 \in \mathcal{S}tate_S(C_2) : o_2(Id) = oid \wedge o_1(Id) \in o_2(r_2).\end{aligned}$$

- Additionally, all constraints have to be satisfied by a database state.

[5] A reference can be restricted to a set containing at most one object identifier by a specific cardinality constraint.

[6] In Sect. 2 we used $I(S)$ to denote the state.

The data model GIM is designed to enable data lossless schema translations from typical database models, e.g. the relational model as well as object-oriented ones.

## 4.2 Graphical representation of a GIM schema

For illustrating GIM schemata, we use a simple 2-dimensional graphical representation. The vertical dimension is defined by the *intension*, i.e. the (typed) attributes including reference attributes. Formally, this dimension is given by $\bigcup_{C \in S.\mathcal{C}} dom(C.Att)$. The type of a reference to extension `n` is denoted with '`En`'.

The horizontal dimension is defined by the set of GIM classes, which are called *base extensions*. These extensions are mutually disjoint and are used as placeholders for concrete objects. For simplicity, base extensions are often identified by numbers.

Figure 9 shows the two-dimensional diagram defining a GIM schema. The GIM schema of our example schema 1 is depicted in Fig. 10 on page 487. References are denoted by a line between the two paired inverse reference attributes. A class is represented by a rectangle grouping attributes for one base extension. The figure shows additionally the representation of local constraints and global

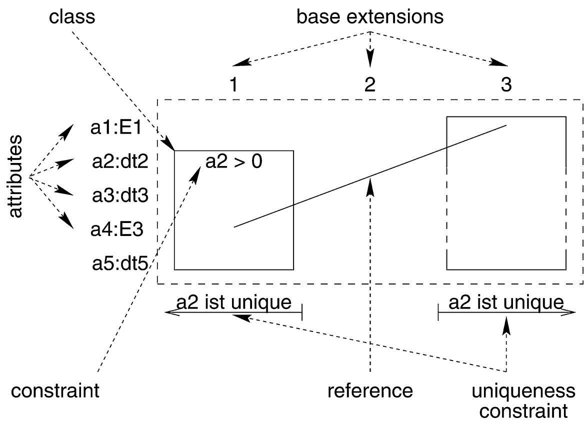


**Fig. 9** Graphical representation of a GIM schema.

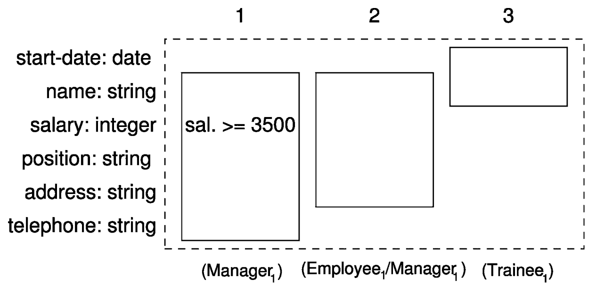


**Fig. 10** Graphical GIM-representation of schema 1.

uniqueness constraints. In Fig. 9 there is a global uniqueness constraint on `a2` ranging base extensions `1` and `3`.

Later on we will use tables as a further simplified notation. Such a table consists of attribute names, numbered base extensions, '✓'-entries for the attribute-extension relation, and shadowed areas for integrity constraints.

The goal of the graphical representation is to demonstrate single integration steps. Of course, only in small schemata every class can group its attributes without gaps by means of rectangles. Large GIM schemata should be better represented using the table notation.

## 5 Schema translation into GIM

After defining the Generic Integration Model, we will discuss the translation of local schemata into GIM. A data lossless translation is feasible for schemata of various source database models. Of course, the precise translations steps depend on the source database model. Due to space limitations, we cannot give a formal description of the translation steps for every possible source database model. Instead, we describe the translation w.r.t the following aspects informally:

1. object identifiers,
2. bidirectional references,
3. atomic data values,
4. integrity constraints, and
5. disjoint class extensions and consequences.

*Object identifiers:* Since the Generic Integration Model supports the idea of objects, new object identifiers must be assigned to tuples (objects). In order to guarantee a bijective database mapping, the mapping between original objects and assigned object identifiers must be stored and maintained by the FDBS.

*Bidirectional references:* A reference value in GIM contains the object identifiers of the referenced objects. Referential integrity is guaranteed by GIM. If the source data model is the relational model then foreign key attributes pointing to primary key attributes are converted into reference attributes. Since GIM requires bidirectional reference attributes, corresponding inverse reference attributes must be created, if necessary, for every class a given reference attribute refers to. On data level, for each reference value from one object to another object an inverse reference value is created and assigned to the corresponding reference attribute. The consistency of this kind of redundancy is enforced by the Generic Integration Model. In our example the attribute `job` of the class $\texttt{People}_2$ is a unidirectional reference attribute pointing to the class $\texttt{Position}_2$. The new reference attribute is the attribute `people` of the class $\texttt{Position}_2$ pointing to class $\texttt{People}_2$.

Please note, that creating object identifiers together with a bijective mapping as well as creating inverse reference attribute values bijectively to original reference attribute values obeys the requirement for schema equivalence. These operations are comparable with the *node creation $\varsigma$-transformation* in [48].

*Atomic data values:* GIM requires the 1NF-attributes. If an input schema is not in 1NF then the relational normalization theory [81] gives us a formal framework to transform it into a normalized schema. Our example schemata are already normalized and need not to be transformed.

**Table 3** S1 and S2: extensional mapping

| local class | base extension |
|---|---|
| $\text{Person}_1$ | 1,2,3 |
| $\text{Employee}_1$ | 1,2 |
| $\text{Trainee}_1$ | 3 |
| $\text{Manager}_1$ | 1 |

| local class | base extension |
|---|---|
| $\text{Position}_2$ | 1 |
| $\text{People}_2$ | 2,3,4 |
| $\text{Customer}_2$ | 2,3 |
| $\text{Employee}_2$ | 3,4 |
| $\text{Cust-Emp}_2$ | 3 |

*Integrity constraints:* In order to guarantee schema equivalence, integrity constraints of a source schema are mapped to GIM constraints. Since GIM supports uniqueness constraints and domain constraints, no semantics gets lost if the source data model is the relational model. In contrast to atomic reference attributes in the relational model, an object model can allow a set of object references for one reference attribute. Therefore, GIM supports the set semantics together with cardinality constraints capable of restricting such sets. In this way, schema equivalence can be reached for translating schemata from the relational model as well as from an object model.

*Disjoint class extensions and consequences:* Class extensions of a GIM schema are always mutually disjoint. This property is usually not fulfilled in other data models. Specializations in object-oriented database models, for example, define subset relations between class extensions. Specialization can also exist in relational schemata specified by using primary and foreign keys. A decomposition of class extensions may be necessary to generate disjoint extensions. For subset relationships in specialization hierarchies, it is sufficient to compute *shallow class extensions*. A shallow extension of a class consists of those instances which are not instances of any of its subclasses. Since reunifying the disjoint extension would reconstruct the original extension, the decomposition obeys the requirement for data lossless schema equivalence.

In a source schema a reference attribute can point to a class which has to be decomposed due to the requirement for disjoint class extensions. Thus, after decomposition this attribute has to refer to more than one class. This is the reason why GIM allows a reference attribute referring to more than one class.

Class decomposition must be in accordance to specified integrity constraints. Domain constraints are directly adopted to the resulting classes. Intra-class uniqueness constraints, however, become inter-class constraints.

The translation of our example schemata provides the GIM schemata presented in Fig. 10 and 11.[7] These GIM schemata in the table notation are shown in Table 4. The mappings between the original classes and the GIM classes are presented in Table 3. Some GIM extensions are expressed by set differences between local class extensions.

[7] Since $\texttt{Employee}_1 \cup \texttt{Trainee}_1 = \texttt{Person}_1$ and $\texttt{Employee}_2 \cup \texttt{Customer}_2 = \texttt{People}_2$ hold (see Sect. 3) the shallow extensions of $\texttt{Person}_1$ and $\texttt{People}_2$ are always empty.

**Table 4** S1 and S2 as GIM tables

| local class | 1 | 2 | 3 |
|---|---|---|---|
| start-date | | | ✓ |
| name | ✓ | ✓ | ✓ |
| salary | ▨ ✓ | ✓ | |
| position | ✓ | ✓ | |
| address | ✓ | ✓ | |
| telephone | ✓ | | |

| local class | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| address | | ✓ | ✓ | |
| name | | ✓ | ✓ | ✓ |
| job | | ✓ | ✓ | ✓ |
| salary | | | ▨ ✓ | ▨ ✓ |
| description | ✓ | | | |
| qualification | ✓ | | | |
| people | ✓ | | | |

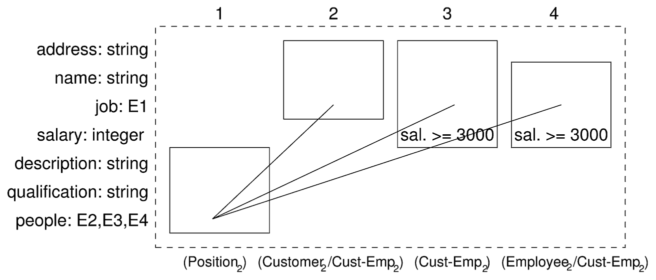


**Fig. 11** Graphical GIM-representation of schema 2.

## 6 Schema homogenization

This section describes how to homogenize the component schemata $S_1^C$ and $S_2^C$ informally. Homogenizing these schemata produces the GIM schemata $S_1^H$ and $S_2^H$. Different classes of conflicts must be considered. A good integration method guides the designer to resolve conflict classes in a particular sequence avoiding redundant integration steps. The GIM method resolves the main conflict classes in the following sequence: (i) *Structure conflicts:* correspondences between attributes and classes due to correspondences between attribute values and objects; (ii) *Attribute conflicts:* correspondences between attributes due to correspondences between their attribute values; and (iii) *Extensional conflicts:* correspondences among classes due to potential redundancy among their extensions.

The list does not explicitly include name conflicts between attributes and classes, respectively, e.g. synonyms and homonyms. Resolving attribute-name conflicts is implied by the resolution of attribute conflicts, and resolving class-names conflicts is implied by the resolution of extensional conflicts.

### 6.1 Homogenization: intensional aspects

As introduced earlier, the intension of a class denotes its defined attributes. Intensional conflicts are, therefore, conflicts which involve attributes. This subsection describes how to deal with structure and attribute conflicts.

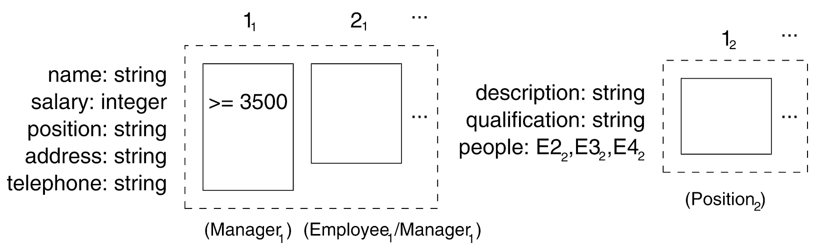


**Fig. 12** Structure conflict between the example schemata.

### *6.1.1 Structure conflicts*

A structure conflict exists if one or more attributes of a schema have a semantic correspondence to a class of another schema. In other words, an attribute value would appear as an object in the other database. In our example a structure conflict exists between the attribute `position` of the classes $1_1$ (`Manager`$_1$), $2_1$ (`Employee`$_1$\\`Manager`$_1$) and the class $1_2$ (`Position`$_2$) (cf. Fig. 12).

To find all structure conflicts the designer has to compare attributes with classes. Attribute and class names are often giving hints to detect structure conflicts. Therefore, a tool which computes the affinity of names can help to detect structure conflicts (see for example [5]).

Due to the demand for minimality and understandability, each concept should exist in only one representation. In general, there are two ways of resolving a structure conflict: the transformation of the attribute into a class or vice versa. The transformation of a class into an attribute is often not feasible because not every class can be transformed to single attributes without data loss. For example, the existence of the attribute `qualification`$_2$ makes the transformation of the class `Position`$_2$ into an 1NF-attribute impossible. Therefore, the GIM method resolves the structure conflict by transforming attributes into classes.

*Schema transformation:* Here we assume the attibutes $A_1 \ldots A_n$ of the classes $C_1 \ldots C_n$ of the first schema with $Dom_{C_1}^{S_1^C}(A_1) = \cdots = Dom_{C_n}^{S_1^C}(A_n)$ are in conflict with the class C of the second schema. For resolving the structure conflict a new class $C_{\text{new}}$ for the first schema has to be created. The name of the new class is adopted from the class C. The class $C_{\text{new}}$ has exactly one non-reference attribute $A_{\text{new}}$ with the name of the corresponding, identifying attribute of class C. The data type is adopted from the attributes $A_1 \ldots A_n$. An additional attribute is a reference attribute $A_{\bar{R}}$ pointing to the classes $C_1 \ldots C_n$. The original attributes $A_1 \ldots A_n$ are now transformed to reference attributes inverse to $A_{\bar{R}}$ referring to the class $C_{\text{new}}$.

In order to guarantee a surjective database mapping integrity constraints have to be considered. Since the semantics of the original attributes is now shifted to attribute $A_{\text{new}}$, corresponding integrity constraints are copied to the new class $C_{\text{new}}$. Additionally, a uniqueness constraint on attribute $A_{\text{new}}$ guarantees that each value of the original attributes corresponds to at most one object of the new class. A null value of the original attributes $A_1 \ldots A_n$ is replaced by a null reference. Therefore, the attribute $A_{\text{new}}$ must be a not-null-attribute. Furthermore, the class $C_{\text{new}}$ can contain only such objects for which corresponding values of the original attributes

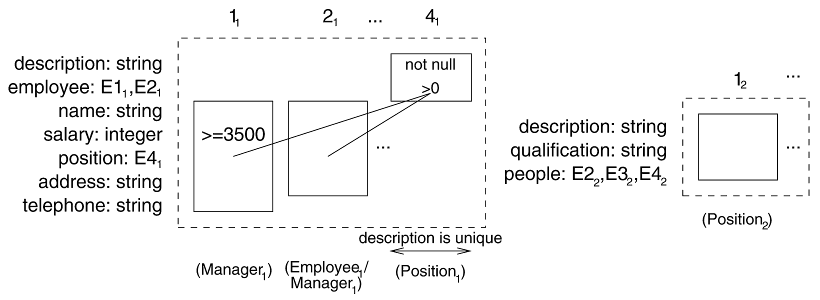


**Fig. 13** Resolution of a structure conflict.

exist. Therefore, each object of the class $C_{new}$ has to have at least one reference, i.e. a not null and a cardinality constraint are defined on the reference attribute.

The resolved structure conflict of our example is depicted in Fig. 13.

*Mapping database states:* For each occurring value of an attribute being involved in a structure conflict a new object must be created and inserted into the corresponding new class. To guarantee schema equivalence the mapping between the object identifier for the created objects and the related attribute values must be bijective. A stored mapping table relates attribute values to object identifiers. Furthermore, the reference attributes must be set to the correct object identifiers.

### *6.1.2 Attribute conflicts*

An attribute conflict between two attributes exists if they represent the same property of objects in different ways. The attributes are semantically related by their related values.

For example, the two attributes `price` and `cost` constitute an attribute conflict. Although the attribute names are different, they denote the same semantics. The conflict can also be on data level, if the price is given USD and the cost in EURO.

Our introduced example contains an attribute conflict between the attributes $\texttt{name}_1$ and $\texttt{name}_2$. Both attributes denote names of persons. The attribute $\texttt{name}_1$, however, contains the last name whereas $\texttt{name}_2$ contains the first and the last name of a person.

In general, the designer has to use background knowledge to detect attribute conflicts. Investigating design documents and current database states can often aid the designer to determine the semantics of attributes. Furthermore, similar to the structure conflict, computing the affinity of attribute names by using a synonym dictionary can give hints for attribute conflicts (cf. [5]). Due to the demand for minimality and understandability the problem of different representations of one property must be resolved.

In this work we only sketch the idea of attribute conflict resolution. For a deeper discussion we refer to [68].

We distinguish between two types of attribute conflicts. In conflicts of type (1), attributes have different precisions. For example, the attribute `name` containing

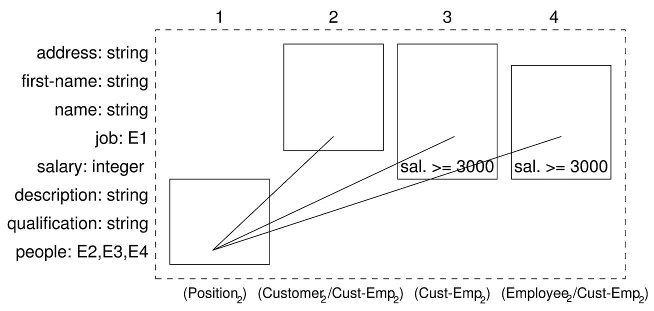


**Fig. 14** Resolution of an attribute conflict in schema 2.

the full name is more precise than just the last name. Another example: one integer attribute specifies the length in meter and the other one in centimeter resulting in non-injective mappings between corresponding values. In order to resolve this type of conflict we split the attribute with higher precision into two attributes. `name` is split into attributes containing first names and last names, respectively.

A bijective mapping between the high-precision attribute values and the value combination of the new attributes must be established in order to guarantee schema equivalence.

Attribute conflicts of type (2) are characterized by different attribute values. Two attributes can, for example, contain names of colors and the first one uses color values in English and the other one in German. Such a conflict is resolved by applying tables which relate corresponding attribute values. For schema equivalence the table must establish a bijective mapping. Appropriate domain constraints must guarantee functionality and surjectivity of the value mapping.

If we regard attributes as nodes in the SIG-formalism of [48] then our proposed schema transformation corresponds to a node creation $\varsigma$-transformation followed by a node deletion $\varsigma$-transformation.

In our example, we have an attribute conflict type (1) between $\texttt{name}_2$ and $\texttt{name}_1$. We split the attribute $\texttt{name}_2$ into attribute $\texttt{name}_2$ and $\texttt{first-name}_2$ which contain last and first names, respectively. Figure 14 depicts the modified schema.

## 6.2 Homogenization: extensional aspects

The Generic Integration Model requires disjoint class extensions. Whereas class extensions within one component database already meet this requirement[8] this is not always the case if we compare class extensions from different databases. This aspect is the topic of this subsection.

The goal is to have identical or disjoint class extensions w.r.t. the schemata $S_1^C$ and $S_2^C$ only. As a prerequisite we need correct information about set relations among class extensions. Extensional relations can be regarded as global integrity constraints hidden in application semantics. Specifying extensional relations is usually very hard for a designer but inevitable for a correct handling of

[8] Disjoint class extensions result from the translation step described in Sect. 5.

redundancy. Such relations can be expressed by an Euler/Venn-diagram or by an extension diagram (cf. Sect. 3). Different sources of information about extensional relations are:

1. *Information from the given schema and data model:* From the specification of an object-oriented schema we have the information which class is a subclass of another class. Such a specialization means a subset relation between their extensions. Furthermore, additional constraints can state exclusive or total specializations.
2. *Database states:* The designer often has access to the states of the databases to be integrated. Comparing available database states helps to detect redundancy. Detected redundancy between two classes excludes disjointness and can give hints for a set equivalence or a subset relation.
3. *Integrity constraints:* Comparing integrity constraints can help to detect extensional relations among classes from different schemata. If no database state can simultaneously fulfill the conjunctively combined integrity constraints of different classes, then a disjoint extensional relation can be concluded. For more details see [55,79,80].
4. *Background knowledge of the designer:* In practical scenarios, information originating from the previous three sources is often not sufficient for a complete extensional analysis. The designer usually has to use his background knowledge in order to specify complete information about extensional relations.
5. *default:* If no information about the extensional relation between classes from different databases is known then we assume per default an overlapping relation.

A prerequisite for resolving extensional conflicts is the existence of an extension diagram. It is often very hard for a designer to specify such a diagram because it relates the extensions of all classes simultaneously. Instead of an extension diagram, the designer usually wants to specify the extensional relation among a *restricted* set of classes. Most publications about schema integration, e.g. [75], propose *extensional assertions*.

An extensional assertion specifies an extensional relation between two extensions. It refers to a stable set relation between database states at any instant of time including the future.

The symbols $\varnothing$, $\subseteq$, $\equiv$, and $\Cap$ denote four different extensional relations between two extensions. The following formalization uses the predicate SAME as introduced in Sect. 1. Between two classes $C_1$ and $C_2$ the following extensional assertions can be formulated:

$$\textit{disjointness}: C_1 \varnothing C_2 :\Leftrightarrow \forall t : \forall id_1 \in Ext^t_{Id,C_1} :$$
$$\forall id_2 \in Ext^t_{Id,C_2} : \neg\text{SAME}(id_1, id_2)$$
$$\textit{containment}: C_1 \subseteq C_2 :\Leftrightarrow \forall t : \forall id_1 \in Ext^t_{Id,C_1} :$$
$$\exists id_2 \in Ext^t_{Id,C_2} : \text{SAME}(id_1, id_2)$$
$$\textit{equivalence}: C_1 \equiv C_2 :\Leftrightarrow (C_1 \subseteq C_2) \wedge (C_2 \subseteq C_1)$$
$$\textit{overlap}: C_1 \Cap C_2 :\Leftrightarrow \neg(C_1 \varnothing C_2) \wedge \neg(C_1 \subseteq C_2) \wedge \neg(C_2 \subseteq C_1)$$

Two classes are extensionally disjoint, if there are no SAME objects in the extensions of these classes at any instant of time. One class is extensionally contained in another class if the extension of the first class is a subset of the extension of the other class at any instant of time. Equivalent classes have same extensions at any instant of time. Finally, an overlap is the remaining case.

In Sect. 3, we motivated the need for extensional assertions among more than two classes. Therefore, we enhance extensional assertions to allow comparisons between *set expressions* on class extensions using the set operations union, difference and intersection (cf. [69]).

The extensional relations for our example are informally given in Sect. 3. The following list contains the explicit extensional assertions:

$\mathtt{Employee}_1 \subseteq \mathtt{Person}_1, \mathtt{Manager}_1 \subseteq \mathtt{Employee}_1,$
$\quad \mathtt{Trainee}_1 \subseteq \mathtt{Person}_1,$
$\mathtt{Customer}_2 \subseteq \mathtt{People}_2, \mathtt{Employee}_2 \subseteq \mathtt{People}_2,$
$\quad \mathtt{Cust\text{-}Emp}_2 \subseteq \mathtt{Customer}_2,$
$\mathtt{Cust\text{-}Emp}_2 \subseteq \mathtt{Employee}_2,$
$\mathtt{Person}_1 \equiv (\mathtt{Employee}_1 \cup \mathtt{Trainee}_1), \mathtt{Employee}_1 \varnothing \mathtt{Trainee}_1,$
$\mathtt{People}_2 \equiv (\mathtt{Customer}_2 \cup \mathtt{Employee}_2), \mathtt{Cust\text{-}Emp}_2$
$\quad \equiv (\mathtt{Customer}_2 \cap \mathtt{Employee}_2),$
$\mathtt{Employee}_2 \subseteq \mathtt{Employee}_1,$
$(\mathtt{Employee}_1 \cap \mathtt{Customer}_2) \subseteq \mathtt{Employee}_2,$
$\mathtt{Position}_2 \varnothing \mathtt{Person}_1, \mathtt{Position}_2 \varnothing \mathtt{People}_2,$

The assertions above are defined on local classes but not on current GIM schemata. Table 3 on P. 489 contains information how the local classes are mapped to GIM classes. Considering these mappings we obtain the following extensional assertions by substitutions:

$$1_1 \cup 2_1 \subseteq 1_1 \cup 2_1 \cup 3_1 \quad (1)$$
$$1_1 \subseteq 1_1 \cup 2_1 \quad (2)$$
$$3_1 \subseteq 1_1 \cup 2_1 \cup 3_1 \quad (3)$$
$$2_2 \cup 3_2 \subseteq 2_2 \cup 3_2 \cup 4_2 \quad (4)$$
$$3_2 \cup 4_2 \subseteq 2_2 \cup 3_2 \cup 4_2 \quad (5)$$
$$3_2 \subseteq 2_2 \cup 3_2 \quad (6)$$
$$3_2 \subseteq 3_2 \cup 4_2 \quad (7)$$
$$1_1 \cup 2_1 \cup 3_1 \equiv 1_1 \cup 2_1 \cup 3_1 \quad (8)$$
$$1_1 \cup 2_1 \varnothing 3_1 \quad (9)$$
$$2_2 \cup 3_2 \cup 4_2 \equiv 2_2 \cup 3_2 \cup 3_2 \cup 4_2 \quad (10)$$
$$3_2 \equiv (2_2 \cup 3_2) \cap (3_2 \cup 4_2) \quad (11)$$
$$3_2 \cup 4_2 \subseteq 1_1 \cup 2_1 \quad (12)$$
$$(1_1 \cup 2_1) \cap (2_2 \cup 3_2) \subseteq 3_2 \cup 4_2 \quad (13)$$
$$1_2 \varnothing 1_1 \cup 2_1 \cup 3_1 \quad (14)$$
$$1_2 \varnothing 2_2 \cup 3_2 \cup 4_2 \quad (15)$$

**Table 5** Extension diagram for $S_1^C$ and $S_2^C$

| $S^C$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $1_1$ | ✓ | ✓ | ✓ | | | | | | | | |
| $2_1$ | | | | ✓ | ✓ | ✓ | | | | | |
| $3_1$ | | | | | | | ✓ | ✓ | | | |
| $4_1$ | | | | | | | | | | ✓ | ✓ |
| $1_2$ | | | | | | | | | | | ✓ |
| $2_2$ | | | | | | | ✓ | | ✓ | | |
| $3_2$ | | | ✓ | ✓ | | | | | | | |
| $4_2$ | | ✓ | | | ✓ | | | | | | |

Resolving structure and attribute conflicts typically changes the involved schemata. In our example a new class $\texttt{Position}_1$ ($4_1$) was created. It must be compared with other classes:

$$\texttt{Position}_1 \varnothing \texttt{Person}_1 : 4_1 \varnothing 1_1 \cup 2_1 \cup 3_1 \quad (16)$$

$$\texttt{Position}_1 \varnothing \texttt{People}_2 : 4_1 \varnothing 2_2 \cup 3_2 \cup 4_2 \quad (17)$$

$$\texttt{Position}_2 \subseteq \texttt{Position}_1 : 1_2 \subseteq 4_1 \quad (18)$$

Since schema translation already performed an extensional decomposition assertions from (1) to (8) and (10) are meaningless. All classes within one GIM schema are per definition mutually disjoint. Therefore, assertions (9), (11), (15), and (16) are dropped.

The next step is the derivation of an extension diagram from a set of extensional assertions. [69] describes an appropriate algorithm for that problem. The algorithm computes an extension diagram with a minimal number of base extensions by exploiting simplification rules of propositional logic. The idea is to transform assertions into boolean expressions, to combine them conjunctively, to transform the result into the disjunctive normal form, and to minimize the number of min-terms. Table 5 shows the computed extension diagram for our example.

*Schema transformation:* An extension diagram defines a set of base extensions. Each component class extension is expressed by the union of corresponding base extensions. Resolving extensional conflicts means extensionally decomposing classes into smaller classes. In case of redundancy a base extension causes the generation of two new classes for both schemata. These classes have the same extension. The name of a new class is set to the number of the corresponding base extension. After splitting we obtain a new set of GIM classes for two schemata. Classes from different schemata with same names always have the same extension, otherwise, their extensions are always disjoint.

As result we obtain the homogenized schemata $S_1^H$ and $S_2^H$. The homogenized schemata of our example are presented in Table 6 and 7. The shaded areas indicate the existence of integrity constraints (cf. Table 8).

*Mapping database states:* Mapping database states for resolving extensional conflicts is similar to the mapping for schema translation. In contrast to the translation, however, splitting classes in smaller classes requires comparisons among

**Table 6** Example: $S_1^H$

| Local class | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| `description` | | | | | | | | | ✓ | ✓ |
| `employee` | | | | | | | | | ✓ | ✓ |
| `start-date` | | | | | | | ✓ | ✓ | | |
| `name` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| `salary` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| `position` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| `address` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | |
| `telephone` | ✓ | ✓ | ✓ | | | | | | | |

**Table 7** Example: $S_2^H$

| Local class | 2 | 3 | 4 | 5 | 7 | 9 | 11 |
|---|---|---|---|---|---|---|---|
| `address` | | ✓ | ✓ | | ✓ | ✓ | |
| `first-name` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| `name` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| `job` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| `salary` | ✓ | ✓ | ✓ | ✓ | | | |
| `description` | | | | | | | ✓ |
| `qualification` | | | | | | | ✓ |
| `people` | | | | | | | ✓ |

**Table 8** Integrity constraints

| GIM class | Integrity constraint |
|---|---|
| 1,2,3 | `salary` $\geq$ 3500 |
| 2,3,4,5 | `salary` $\geq$ 3000 |
| 10,11 | `description` is not null |
| 10,11 | card (`employee`) > 0 |
| 10∪11 | unique (`description`) |

extensions from different databases. For example, base extension `2` contains all manager objects of the first database which are simultaneously employee but not customer objects of the second database.

The relation SAME relates objects from different databases. This relation is an essential element for splitting class extensions. Here, we assume the existence of such a relation. For producing and managing such a relation we refer to [9, 44, 54, 64, 88].

Splitting a class extension into disjoint class extensions respects schema equivalence. A simple union on the generated classes creates the original class extension without loss of data.

## 7 Schema merging

This design step merges the homogenized schemata $S_1^H$ and $S_2^H$ into the integrated schema $S^I$. In schema $S^I$, class names should be unique. Between the

**Table 9** Example – integrated schema

| Local Class | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| `first-name` | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | |
| `name` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| `salary` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | |
| `position` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | |
| `address` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | | |
| `telephone` | ✓ | ✓ | ✓ | | | | | | | | |
| `start-date` | | | | | | | ✓ | ✓ | | | |
| `job` | | ✓ | ✓ | ✓ | ✓ | | ✓ | | ✓ | | |
| `description` | | | | | | | | | | ✓ | ✓ |
| `employee` | | | | | | | | | | ✓ | ✓ |
| `qualification` | | | | | | | | | | | ✓ |
| `people` | | | | | | | | | | | ✓ |

homogenized schemata equal class names indicate equal extensions. Classes without an equivalent[9] class in the other schema are adopted into the integrated schema. Pairs of equivalent classes, however, are merged into one class, respectively. A class $C_1 \in S_1^H.\mathcal{C}$ and a class $C_2 \in S_2^H.\mathcal{C}$ with $C_1.Cname = C_2.Cname$ are merged into $C \in S^I.\mathcal{C}$ with $C.Cname = C_1.Cname$ and $C.Att = C_1.Att \cup C_2.Att$.

An interesting question is how to merge integrity constraints. Different sets of integrity constraints can restrict equivalent classes even on same attributes. Reasons for differences are usually: (i) *Incomplete design:* The specified constraints are too weak in one schema or some constraints have been simply forgotten; and (ii) *Wrong homogenization:* Conflicting integrity constraints can indicate a wrong homogenization. For example, wrong attributes are related to each other or some class extensions are specified to overlap although they are disjoint.

Following [3,14,20,21], the detection of conflicting integrity constraints is in general an undecidable problem. Restricting the expressiveness power of integrity constraints, however, makes the problem decidable (cf. [80]). In a GIM schema, only basic integrity constraints are supported. We assume the designer is able to detect conflicting integrity constraints and to find the reasons for them. A wrong homogenization forces a redo of the homogenization. If, however, conflicts are caused by incomplete designs then the different sets of integrity constraints are conjunctively combined for the integrated class. The combination is a conjunction since each instance of one class has always a SAME object in the corresponding class. If the object states are correct then they must fulfill the integrity constraints of both classes simultaneously:

$$C.IC = C_1.IC \cup C_2.IC$$
$$S.Unique = S_1^H.Unique \cup S_2^H.Unique$$

Merging the homogenized schemata of our example produces an integrated schema presented in Table 9.

[9] Two classes are equivalent if they have the same name and thereby the same extension.

*Mapping database states:* A mapping problem occurs due to semantic redundancy: The redundant values of SAME objects should always be equal but in practice they often differ. Various reasons can cause different values: (i) *Timeliness:* Applications update values with different delays after a real-world object has been changed. In other words, different values reflect real-world values at different instants of time. (ii) *Wrong data:* Often, a value does not correctly reflect an attribute value of a real-world object. Reasons are, for example, misspellings, imprecise measurements, and badly chosen attribute domains. (iii) *Wrong homogenization:* Different values can be caused by wrong resolutions of attribute or extensional conflicts.

Mapping database states with respect to redundant data means mapping a pair of attribute values to exactly one value for the integrated schema. Such a mapping, in general, depends on the semantics of the corresponding attributes and their applications. Therefore, there is no standard solution to how to find out the correct value. For each attribute A occurring in equally named classes the designer has to define a function $\gamma_A$:

$$\gamma_A \subseteq \left(Dom_C^{S_1^H}(A) \times Dom_C^{S_2^H}(A)\right) \times Dom_C^{S^I}(A)$$

Depending on the quality of data, cf. [29], the function has to assign values to pairs of given values. The following list shows some possible ways of computation: (i) *First (or second) value:* The function takes always the first (or second) value since the designer knows that the first (second) database contains more correct values. (ii) *Average value:* Under some circumstances the average value should be presented in the integrated schema. (iii) *Maximum (minimum) value:* Sometimes the designer knows, that the maximum (minimum) of both values should be chosen. (iv) *Dependency:* The decision on the right value can depend on other attributes. If, for example, a time attribute carries a time stamp of the last update, then both time stamps need to be compared to find the most recent value. If the input values of $\gamma$ are equal, then the output value should be that value.

In general, the function $\gamma$ is not injective and causes therefore a data loss. If the designer does not want a loss of data then she/he must adopt both attributes as being unrelated.

For a non-injective function, no inverse function can exist. In order to map an inserted global value to the underlying databases, however, an inverse function is required. The commonly used function is the identity function. More precisely, every value ‘x’ is mapped to the pair ‘(x, x)’. That is, the global value is inserted into the underlying databases.

Besides mapping attribute values, object identifiers have to be mapped, too. The local object identifier cannot be used directly on the global level. Following the approach in [64] a bijective mapping between local and global mapping is required.

The mapping function is a total and surjective function. Due to the missing injectivity of the functions $\gamma$ the mapping is not injective, too. However, with respect to non-redundant data the mapping guarantees schema equivalence. Merging equivalent class extensions with different attributes and a bijective SAME relation mapping between them corresponds in the SIG-formalism to moving edges to one side of bijectivley related nodes ($o$-transformation) followed by removing a node (node deletion $\varsigma$-transformation).

## 8 External schemata derivation

An integrated GIM schema $S^I$ is not an appropriate schema for applications due to the GIM modelling restrictions. Instead of being a schema for applications, it allows a very elegant derivation of external schemata. If we look at an integrated GIM schema, e.g. Table 9, we can make some interesting observations on which the derivation algorithm of external schemata bases:

- *Rectangles represent classes:* In the schema you can find rectangles completely filled with checkmarks. A rectangle means here that for a certain set of base extensions all values for a subset of attributes are available. The form of a rectangle is dependent on the given order of columns and rows. These orders, however, are not meaningful. Therefore, we speak of a rectangle if there is a rectangle in at least one attribute and base extension order. Rectangles can be regarded as classes. We can find even local classes as rectangles. For instance the local class `People`$_2$ is represented by the rectangle with the base extensions 2–5, 7, and 9 and the attributes `first-name`, `name`, and `job`.
- *There are maximal rectangles:* Some rectangles can be augmented by some base extensions or attributes. Rectangles which cannot be augmented are called *maximal* rectangles. The local class `People`$_2$, for example, can be augmented by the attribute `address` and is therefore not a maximal rectangle.
  Deriving external schemata means finding all maximal rectangles. Due to the demand for completeness each checkmark in the table must be contained in at least one rectangle.
- *Maximal rectangles can overlap:* Fig. 15 shows an overlap of two maximal rectangles. Thereby, if the extension of a class (rectangle) is a subset of the extension of another class then their attribute sets are always in a superset relationship, a so-called Galois connection:

$$Ext(C_1) \subseteq Ext(C_2) \Leftrightarrow Int(C_2) \subseteq Int(C_1).$$

  Due to this observation, we can consider an overlap as a specialization between classes. The class with the smaller extension is the subclass.

We will exploit this observation in the next two subsections. The first subsection shows how to derive a first external schema from an integrated schema. The second subsection explains the adaption of an external schema to a certain application view.

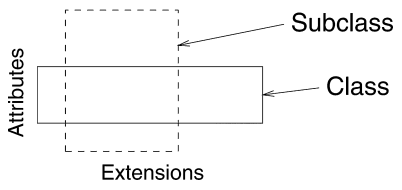


**Fig. 15** Overlapping maximal rectangles.

## 8.1 Automatic derivation of a first external schema

Finding manually all maximal rectangles is very hard for a designer. For automation we propose to apply mechanisms from the theory of formal concept analysis.

### *8.1.1 Formal concept analysis*

The theory of formal concept analysis was developed by mathematicians working in the area of lattice theory. We recommend the book [23] for more information about this theory.

The theory of concept analysis is based on the following formalization [19]: 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. The binary relation $I$ expresses that the object $g \in G$ has the attribute $m \in M$ whenever $(g, m) \in I$ holds.

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

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

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

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

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

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

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

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

As result, we obtain a lattice denoted by:

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

The lattice operations are given by the following definitions:

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

and

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

The lattice built from concepts is called *concept lattice* where ($extent$(M), M) is the infimum and (G, $intent$(G)) is the supremum of all concepts.

### *8.1.2 Concept analysis for deriving external schemata*

The theory of concept analysis can be adapted to solve the problem of deriving external schemata. A GIM-schema assigning attributes to base extensions, e.g. expressed by Table 9, can be interpreted as a binary relation $I$, i.e. it represents a context.

$$G := \{C.Cname | C \in S^I.\mathcal{C}\}$$

$$M := \bigcup_{C \in S^I.\mathcal{C}} dom(C.Att) \setminus \{Id\}$$

$$I := \{(g, m) \in G \times M | \exists C \in S^I.\mathcal{C} : C.Cname = g \wedge m \in dom(C.Att)\},$$

where $g$ corresponds to a base extension and $m$ is an attribute. A concept lattice derived from such a GIM-schema can be regarded as an external schema as follows: (i) a concept $(A, B) \in L$ is a class with extension $A$ and its attributes $B$; (ii) $\leq$ is the specialization relationship between two classes; (iii) $\wedge$ is the specialization operation (intersection of extensions) of two classes; (iv) $\vee$ is the generalization operation (intersection of attribute sets) of two classes; (v) $(extent(M), M)$ is the bottommost class of the specialization hierarchy (attribute set contains all attributes; extension may be empty); (vi) $(G, intent(G))$ is the topmost class of the hierarchy (extension is the union of all base extensions; attribute set may be empty).

Now we can transform the problem of finding maximal rectangles into the theory of concept analysis. Each GIM schema can be regarded as a context. The concept lattice computed from the context can then directly be interpreted as the specialization hierarchy of an external schema. Due to the fact, that each concept represents a *maximal* rectangle, the classes cannot be further extended by any base extension or attribute.

*Computational complexity:* Unfortunately, this approach has two shortages: a context with $m := |G|$ and $n := |M|$ can contain at most $2^{\min(m,n)}$ concepts. Each algorithm for constructing concept lattices from a given context is therefore in the worst case inherently of exponential complexity. Exponential complexity, however, is unacceptable for real-sized problems of schema integration.

*Unnecessary classes:* The approach suffers from a further shortage. It produces some concepts representing unnecessary classes. Consider the example context depicted in Table 10.

The Hasse-diagrams of the resulting concept lattice are depicted in Fig. 16.

**Table 10** Context producing unnecessary classes

| M/G | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| a | | | √ | |
| b | √ | √ | √ | |
| c | | √ | √ | √ |
| d | | √ | | |

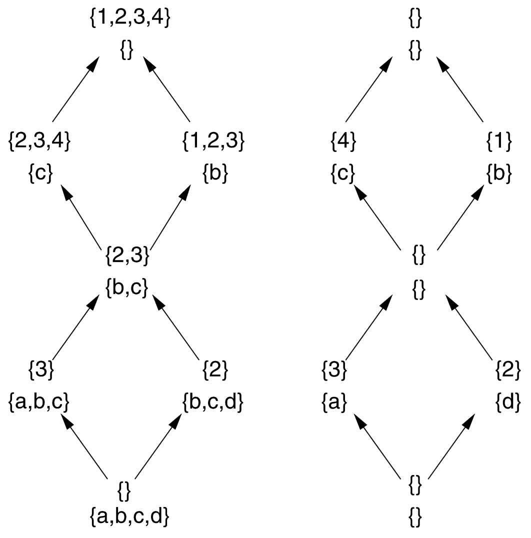


**Fig. 16** Resulting and reduced concept lattice.

The left diagram shows the concepts with the extensional elements and attributes, respectively. Due to the subset relationships between the attribute sets and the extensions, respectively, the left diagram can be reduced to the right diagram without loss of information. The reduced diagram is only another representation of the lattice. Attributes are inherited downwards and extensions are 'inherited' in the opposite direction. The reduced diagram contains three concepts with empty attribute sets and empty extensions. If we regard these concepts as classes then they do not introduce new attributes nor have extensions of their own. In other words, for these classes all attributes are inherited and the extensions are given by the union of their subclass extensions. Such classes are *unnecessary classes* for a database schema and should be omitted.

We will refer to concepts (classes) having an extension or/and attribute of their own as *valid* concepts (classes) otherwise as unnecessary concepts (classes).

That is, given a lattice $\mathcal{L} = (L, \leq, \wedge, \vee, Inf, Sup)$ and a concept $(A, B) \in L$ then let $\mathcal{A} = \{a \in A_i | (A_i, B_i) \in L \wedge A \neq A_i \wedge (A_i, B_i) \leq (A, B)\}$ and $\mathcal{B} = \{b \in B_i | (A_i, B_i) \in L \wedge B \neq B_i \wedge (A, B) \leq (A_i, B_i)\}$. If $\mathcal{A} = A \wedge \mathcal{B} = B$ holds then we call the concept $(A, B)$ an *unnecessary concept*, otherwise a *valid concept*.

From a reduced concept lattice we can easily see that at most $m + n$ *valid* concepts can exist. In that case each concept (class) contains exactly either one object (base extension) or one attribute.

From the discussion above follows an interesting question: *Is there an algorithm to generate all valid concepts (classes) in acceptable time?*

### *8.1.3 The GIM algorithm*

In this subsection we present an algorithm which generates valid classes only. The algorithm computes a concept hierarchy instead of a concept lattice. The algorithm has the computational complexity of $O(n^3)$. For each step of the following algorithm the complexity is 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 input for the algorithm is a context $(G, M, I)$. We use the functions *intent* and *extent* introduced in the previous section to find the sets $Int$ and $Ext$. The sets $Int$ and $Ext$ contain the intents for every single base extension ($g \in G$) and the extents for every single attribute ($m \in M$), respectively:

$$Int := \{intent(\{g\}) \mid g \in G\}$$
$$Ext := \{extent(\{m\}) \mid m \in M\}$$

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

From $Int$ and $Ext$ the two sets $Con_I$ and $Con_E$ containing concepts are derived:

$$Con_I := \{(extent(I), I) \mid I \in Int\}$$
$$Con_E := \{(E, intent(E)) \mid E \in Ext\}$$

The complexity to compute $Con_I$ is $O(n^3)$. The set $Int$ contains at most $n$ elements. Each element of $Int$ 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 $Con_E$ is analogous to $Con_I$.

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

$$Con := Con_I \cup Con_E$$

**Theorem 1** *The class set Con equals the set of* valid *concepts of the corresponding concept lattice.*

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

We proof this proposition by examining the two implications between both sets. The proof refers only to valid concepts with attributes of their own ($(A, B) \in L$ with $\mathcal{B} \neq B$) and to $Con_E$ because any proposition attributed to $M$ holds for $G$, too.

1. *Each valid concept with own attributes corresponds to a class in $Con_E$:*
   A valid concept encompasses 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 $Ext$ and $Con_E$.
   Suppose a valid concept $c$ has the the extension $\{g_1^c, \ldots, g_a^c\}$ and the attributes $\{m_1^c, \ldots, m_b^c\}$. If a concept encompasses more than one attribute then two different cases are possible:
   (a) $\exists m_i^c \in [m_1^c, \ldots, m_b^c] : \forall g \in G :$
   $g \notin [g_1^c, \ldots, g_a^c] \Longrightarrow (g, m_i^c) \notin I$
   The corresponding class is found by computing
   $(extent(\{m_i^c\}), intent(extent(\{m_i^c\})))$ within $Ext$ and $Con_E$.
   (b) $\forall m_i^c \in [m_1^c, \ldots, m_b^c] : \exists g \in G :$
   $g \notin [g_1^c, \ldots, g_a^c] \wedge (g, m_i^c) \in I$

```
boolean [][] context = {{true, false, ...},...};
typedef struct(colid [] cols, rowid [] rows) concept;
concept [] conI, [] conE, [] con;
conI={}; conE={};
rowid [] rows;
colid [] cols;
for each col j ∈ context {
    rows = intent({$j$});
    conI = conI ∪ {(extent(rows),rows)};}
for each row i ∈ context {
    cols = extent({$i$});
    conE = conE ∪ {(cols,intent(cols))};}
con = conE ∪ conI;
```

```
rowid [] intent(colid [] A){
    rowid [] result = {};
    boolean test;
    for each row i ∈ context {
        test=true;
        for each column j ∈ A
            if (context[i][j]==false)
                test=false;
        if (test) result = result ∪ {$i$};
    }
    return result;
}
```

```
colid [] extent(rowid [] B){
    colid [] result = {};
    boolean test;
    for each col j ∈ context {
        test=true;
        for each row i ∈ A
            if (context[i][j]==false)
                test=false;
        if (test) result = result ∪ {$j$};
    }
    return result;
}
```

**Fig. 17** Algorithm to compute *Con*.

Each row $m_i^c$ results in a concept $(extent(\{m_i^c\}), intent(extent(\{m_i^c\})))$ within $Ext$ and $Con_E$. The concepts resulting from each attribute $m_i^c$ have more objects (base extensions) than concept $c$ and are, therefore, superconcepts (superclasses) of concept $c$ with the corresponding attribute $m_i^c$ of their own. The attributes $\{m_1^c, \ldots, m_b^c\}$ of concept $c$ are inherited from its superconcepts. Hence, concept $c$ has no own attributes. That case, however, cannot occur for a valid concept with own attributes.

2 *Each class of $Con_E$ corresponds to a concept with own attributes:*
Suppose the class $c$ of $Con_E$ 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 $intent(extent(\{m\}))$ no superclass can encompass attribute $m_i$.

From both implications follows the proposition. □

Figure 17 gives the algorithm to compute *Con*. The input context is a two-dimensional, Boolean array.

The set *Con* is interpreted as the set of external classes. For a specialization hierarchy we have to compute the inheritance relationships. We build a square matrix *Mat* representing the irreflexive binary relation '<' defined on the generated classes of *Con*.

$$C_1 = (A_1, B_1) \in Con : C_2 = (A_2, B_2) \in Con : C_1 < C_2 \Leftrightarrow A_1 \subset A_2$$

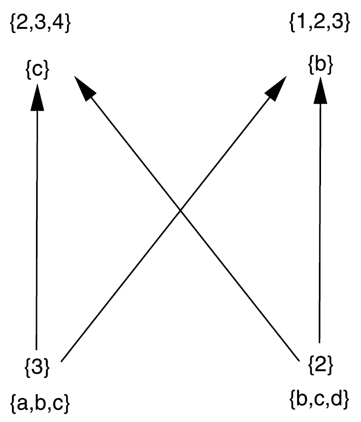


**Fig. 18** Resulting concept hierarchy.

A value '1' in $Mat$ on row $i$ and column $j$ means that class $C_i$ is a subclass of class $C_j$ $(C_i < C_j)$. If no subset relation between class $C_i$ and class $C_j$ exists $(C_i \not< C_j)$ then we write the value '0' into the corresponding matrix field. Complexity to construct this matrix is again $O(n^3)$. Computing the matrix needs comparisons among at most $2 * n$ classes. For each comparison at most $n$ base extensions have to be examined.

The computation $Mat_N = Mat - (Mat \times Mat)$ removes transitive specializations. Each value '1' represents a non-transitive sub/super-class relation. Complexity to multiply matrices is $O(n^3)$. The set $Con$ in combination with the matrix $Mat_N$ gives the external classes with their specializations. Since the computational complexity for every step is not higher than $O(n^3)$ and every step is performed only once the overall complexity is $O(n^3)$.

The algorithm generates a concept hierarchy (cf. Fig. 18) from the context depicted in Table 10.

*Example:* The integrated schema of our example presented in Table 9 on P. 498 is regarded as a context. The GIM algorithm computes the class set $Con = \{\texttt{C1}, \ldots, \texttt{C11}\}$. The extensions and attributes of these classes are presented in Table 11.

### *8.1.4 Schema derivation*

From the class set $Con$ and matrix $Mat_N$ we have information about classes with their extensions and attributes, and about specialization relations among them. However, that is not enough to produce an object-oriented, external schema. The designer has to assign comprehensible names to the classes. A tool can assist this process. When the extension of a new class is the same as the extension of a local class then the name can often be reused.

The derivation algorithm helps us to compute an external schema as an object-oriented one. Of course, in some circumstances the designer wants to derive an external schema in some other data model. In such cases, the set $Con$ and the

**Table 11** Example – global classes

| Class | Extension | Attributes |
|---|---|---|
| `C1` | 1-9 | `name` |
| `C2` | 7,8 | `name`, `start-date` |
| `C3` | 1-7,9 | `name`, `address` |
| `C4` | 2-5,7,9 | `first-name`, `name`, `address`, `job` |
| `C5` | 1-6 | `name`, `address`, `salary`, `position` |
| `C6` | 7 | `first-name`, `name`, `start-date`, `address`, `job` |
| `C7` | 2-5 | `first-name`, `name`, `address`, `job`, `salary`, `position` |
| `C8` | 1-3 | `name`, `address`, `salary`, `position`, `telephone` |
| `C9` | 2,3 | `first-name`, `name`, `address`, `job`, `salary`, `position`, `telephone` |
| `C10` | 10,11 | `description`, `employee` |
| `C11` | 11 | `description`, `employee`, `qualification`, `people` |

table $Mat_N$ are very helpful, too. In the following we will concentrate on object-oriented, external schemata only.

There is a special case where the algorithm produces a loss of data. Therefore, we introduce the idea of a discriminant attribute, which is described in the next paragraph.

*Discriminant:* Sometimes, the proposed algorithm joins together base extensions to external classes in such a way that from an external object we cannot determine its origin, i.e. its base extension. From this situation, a problem arises if this information carries semantics. If, for example, we have the local classes `Man` and `Woman` with the same set of attributes then the algorithm joins the local classes together to one external class. An object of this external class does not have the information about the gender.

In order to avoid this kind of data loss we require that for each object on the external level we are able to determine its base extension from its state and class membership. This property is violated if the algorithm cannot distinguish between base extensions, i.e. if the integrated schema contains base extensions sharing the same attributes. Therefore, we introduce an artificial attribute called the *discriminant* for all base extensions $g \in G$ for which at least one other base extension with same attributes exists:

$$\exists g' \in G : g \neq g' : \forall m \in M : (g, m) \in I \Leftrightarrow (g', m) \in I$$

The idea of the discriminant in the area of schema integration was proposed in [26]. We have adapted this idea to our approach.

The value of the discriminant is the number of the corresponding base extension. On external level we recommend choosing a more meaningful name for the discriminant and mapping the base extension numbers to meaningful values, e.g. `male` and `female` in the mentioned example. A discriminant is also helpful for global insertions of objects. Depending on its value we exactly know the local classes in which an object has to be inserted.

In the integrated schema depicted in Table 9 on P. 498 you will find the base extensions 2 and 3, and the base extensions 4 and 5 sharing the same attributes, respectively. Therefore we introduce a discriminant for these base extensions. The

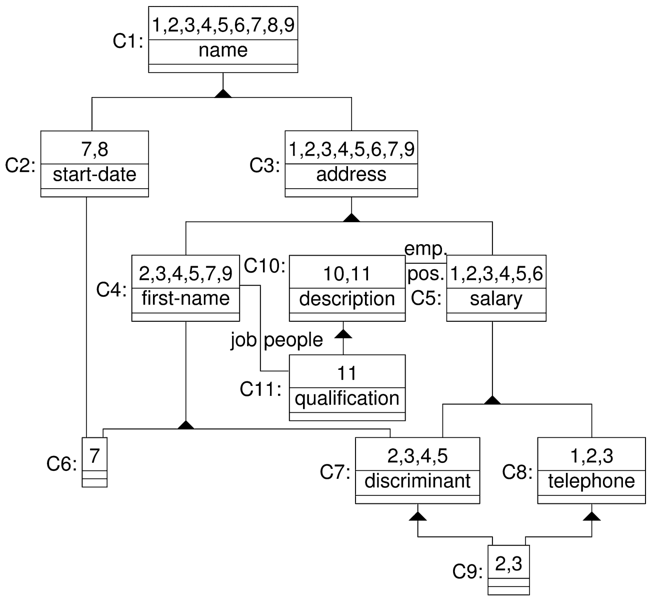


**Fig. 19** Example – first integrated schema.

discriminant determines whether a person is a customer or not. Since the insertion of a new attribute changed the original GIM schema we have to apply our GIM algorithm again which expands the class C7 by the discriminant. The matrix *Mat* remains unchanged. Figure 19 depicts the corresponding external schema without class names but information about the class extensions and attributes.

*Integrity constraints:* During the derivation of an external schema we must consider integrity constraints. Constraints of an external schema restrict insertions of and updates on global objects to objects which can be stored in the underlying databases and fulfill thereby the demand for surjectivity of the corresponding database mapping. If integrity constraints would not be considered then global applications would experience an inconsistent system where a global insertion is rejected due to an invisible local constraint violation. Please remember, a federated database system has to behave as a normal database system for global applications. The publications [11, 13] discuss how to deal with integrity constraints during the schema integration. We explain here our main idea very shortly and informally.

The data model GIM supports two kinds of integrity constraints: attribute domain constraints and uniqueness constraints.

*domain constraints:* If all the base extensions of a derived external class have a domain constraint IC then IC is defined for the external class, too. In an object-oriented schema, similar to attributes, integrity constraints are inherited. If a subclass has the same constraint as the superclass then the constraint can be omitted in the subclass.

Sometimes the involved base extensions of a constraint do not exactly correspond to any external class. In that case this constraint cannot be completely

**Table 12** Integrity constraints

| Class | Integrity constraint |
|---|---|
| C8 | $\texttt{salary} \geq 3500$ |
| C7 | $\texttt{salary} \geq 3000$ |
| C10 | `description` is not null |
| C10 | card(`employee`) > 0 |
| C10 | unique(`description`) |

defined in the external schema. We propose in this case to search the smallest external class which includes all constraint base extensions and to connect the constraint with the corresponding discriminant values:

$$\forall o_1 : o_1(\texttt{discriminant}) \in \{\text{value}_1, \ldots, \text{value}_n\} \Rightarrow \text{IC}$$

*Uniqueness constraints:* If the base extensions of a uniqueness constraint are the same as of an external class then the constraint is defined on this class. Sometimes, such an external class does not exist but the union of some external classes equals this set of base extensions. In that case the constraint can be expressed by an inter-class uniqueness constraint. If this variant also fails then the use of a discriminant attribute is the last solution. We extend a minimal superclass $C$ containing the base extensions of the constraint on attribute A with:

$$\begin{gathered}\forall o_1, o_2 \in C : o_1(\texttt{discriminant}) \in \{\text{value}_1, \ldots, \text{value}_n\} \wedge \\ o_2(\texttt{discriminant}) \in \{\text{value}_1, \ldots, \text{value}_n\} \Rightarrow \\ (o_1(A) = o_2(A) \Rightarrow o_1 = o_2)\end{gathered}$$

In our example the integrity constraints of the base extensions are shown in Table 8. The base extensions of the integrity constraints directly correspond to external classes. The resulting integrity constraints can therefore very easily be assigned to external classes as shown in Table 12.

The derivation of a first, external schema must be accompanied by a bijective database state mapping. The mapping is fixed by the extensional composition of base extensions to external classes. For the correct mapping in the inverse direction we introduced the concept of discriminant and showed how to deal with integrity constraints. The discriminant helps to map every object of the external schemata to exactly one base extension.

## 8.2 Derivation of application views

In the previous subsection we described the derivation of a first, external schema. In correspondence to the 3-level-schema-architecture, cf. [77], external schemata express views for certain applications. One external schema should meet the requirements of a certain application. Usually, the designer of an external schema wants to influence the derivation process corresponding to his certain view.

In our GIM approach the designer starts with the first, external schema. There are many operations to adapt the first schema. The operations are presented in Table 13. The first two categories encompass operations already introduced in

**Table 13** View operations

| Category | Operation |
|---|---|
| translation | complex data type<br>unidirectional reference |
| homogenization | structure operation<br>attribute operation<br>renaming |
| derivation | projection<br>class selection<br>extensional expansion<br>attribute expansion<br>removing classes<br>merging classes |

Sect. 5 and Sect. 6, respectively. The operations of the different categories are explained in the following subsections.

### *8.2.1 Translation and homogenization operations*

The translation operations are operations concerning schema translation when a GIM schema is involved. In contrast to Sect. 5, the derivation of an external schema means performing a translation starting with a GIM schema. The operations include operations to construct complex datatypes as an inverse step to the normalization step. Furthermore, the designer can transform bidirectional references into unidirectional references.

For our example we decide to drop the reference attributes `emp` and `people` from the classes `C10` and `C11`, respectively, because we want to support the inverse directions only. The dropping is done by ignoring the respective rows in the GIM schema.

Of course, depending on the target data model many further operations are possible. Here, we have restricted ourselves to the most common translation operations.

The translation operations are executed in the inverse direction as explained in Sect. 5. The homogenization operations, however, can be performed in both directions. Structure operations, as introduced in Sect. 6.1.1, allow the transformation of attributes into classes and vice versa. Attribute operations, explained in Sect. 6.1.2, encompass composition and decomposition of attributes as well as mapping to new data types and values.

In our example we map the values of the discriminant to boolean values because we must distinguish between base extension 2 and 3 and between 4 and 5, respectively. The values 3 and 4 are mapped to `true` whereas the values 2 and 5 are mapped to `false`. The discriminant indicates whether a person is a customer.

The renaming operation involves attribute and class names. An extensional comparison of external classes with local classes helps to find comprehensible class names. Local attribute names can often be adopted. In our example we give the introduced discriminant the new name `customer`.

### *8.2.2 Derivation operations*

Derivation operations were not introduced yet. The operations `projection`, `class selection`, `extensional expansion`, and `attribute expansion` filter database states. In other words, using these operations the designer decides on a loss of data. Thus, the corresponding database mappings cannot be injective mappings.

– *Projection:* Projection means that not each attribute of the first external schema should be available in the final external schema. Therefore, in the GIM table certain rows are eliminated before our derivation algorithm is applied.
– *Class selection:* Not each base extension should appear in an external class. This can be done by eliminating certain columns from the GIM schema.
– *Extensional and attribute expansion:* Sometimes the designer wants to derive either an external schema as an extensional or an attribute expansion of a particular local schema. For the attribute expansion all base extensions are removed which are not covered by a local schema. For extensional expansion the attribute set is restricted to attributes which appear in a certain local schema.

After performing these operations the derivation algorithm must be applied again.

Schema integration often produces very complex schemata, i.e. the number of external classes is often very high. Although the derivation algorithm computes maximal rectangles in some cases the number of classes can be further reduced. Reducing the number of classes meets the demand for minimality. Depending on the view, the understandability is not always improved as well. Therefore, the designer has to decide on the application of these operations. Removing and merging classes are operations defined on the classes of the first external schema. The following paragraphs introduce three different reduction operations.

*Removing abstract superclasses:* Sometimes the algorithm computes external classes without extensions of their own. The extension $Ext_C$ of such a class $C = (Ext_C, Int_C) \in Con$ equals the union of its subclass extensions:

$$Ext_C = \bigcup \{Ext_{CSub} | (Ext_{CSub}, Int_{CSub}) \in Con \wedge Ext_{CSub} \subset Ext_C\}$$

Such classes are often called *abstract* classes or classes with empty shallow extensions.

For example, the class `C3` of Fig. 19 on P. 508 is an abstract class. An abstract class can be removed without loss of data. Of course, the attributes inherited from the removed superclass must be explicitly made visible in its subclasses and the matrices *Mat* and $Mat_N$ must be recomputed. Removing abstract classes reduces the number of external classes but sometimes it reduces the understandability of an external schema, too. Therefore, the designer must trade off minimality against understandability.

In our example we remove the class `C3`. Furthermore we give the classes meaningful names and obtain the external schema depicted in Fig. 20.

*Removing subclasses:* Similar to abstract classes, an external schema can contain classes without own attributes. In other words, all attributes of a class

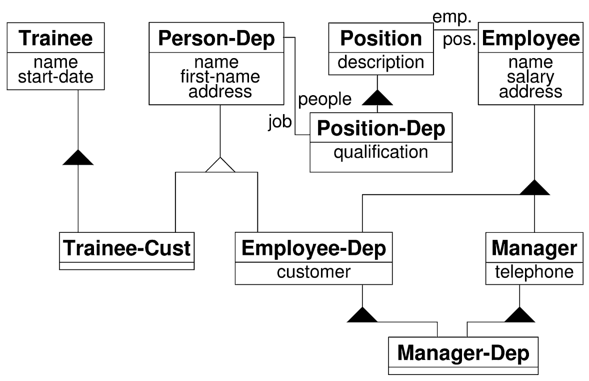


**Fig. 20** Example – external schema.

$C = (Ext_C, Int_C) \in Con$ are inherited:

$$Int_C = \bigcup \{Int_{CSup} | (Ext_{CSup}, Int_{CSup}) \in Con \wedge Int_{CSup} \subset Int_C\}$$

If such a class is removed then its objects belong to more than one most specialized class. A role concept, cf. [30,86], allows such a multiple membership of objects but is often not available in database models. Therefore, classes without attributes of their own can only be removed if the underlying data model supports the role concept.

In our example the classes `C6` and `C9` are such classes.

*Merging of classes:* After removing super- and subclasses the number of external classes can only be further if the designer merges external classes. Such a merging of external classes to one external class is only possible if null values are allowed to appear. If some classes are mutually very similar then null values appear only for few attributes for objects of some base extensions. An extreme situation occurs if one merges all classes into one class which produces a universal relation. Such an extreme situation violates, however, the rules of a good design. Therefore, the designer must very carefully decide on merging external classes.

Suppose, the classes $C_1$ and $C_2 \in Con$ should be merged. Then we obtain the intension and extension of the resulting external class as follows:

$$Int := Int_{C_1} \cup Int_{C_2}$$

$$Ext := Ext_{C_1} \cup Ext_{C_2}$$

Null values appear on attributes from $Int_{C_1} \setminus Int_{C_2}$ for objects from $Ext_{C_2} \setminus Ext_{C_1}$ and vice versa.

For each attribute from $Int$ and for each base extension from $Ext$ a checkmark must be set in the integrated schema. As the next step a discriminant attribute must be introduced for the base extensions of $Ext$. An integrity constraint restricts in dependence on the discriminant the occurrence of null values for updated or inserted global objects. A new application of the derivation algorithm then generates the new external schema.

For instance, the designer may decide to merge the classes `Position` and `Position-Dep`. The unions of the intensions and extensions, respectively, are:

$$Int = \{\texttt{description, qualification}\}$$
$$Ext = \{10, 11\},$$

In the GIM table (Table 9 on Page 498) a checkmark must be set for the base extensions 10 and the attribute `qualification`. The discriminant must then be defined for base extensions ranging from 1 to 5. An additional integrity constraint requires a null value for objects from base extension 1 for the attribute `first-name` and `job`, and for objects from base extensions 4 and 5 for attribute `telephone`.

The resulting final schema is depicted in Fig. 21 on Page 513. For comparison we show the two input schemata again in Fig. 22. Table 14 shows the extensional mapping of local classes to external classes and Table 15 the mapping in the opposite direction using the discriminant. Such mappings can be automatically derived from the mappings of the local schemata to the integrated schema and from the further mappings up to the external schema.

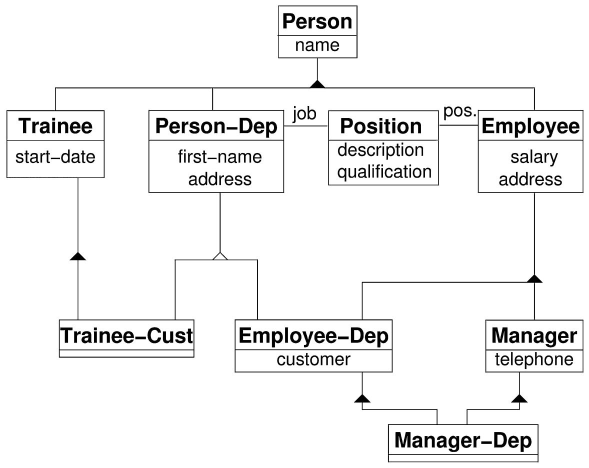


**Fig. 21** Example – final schema.

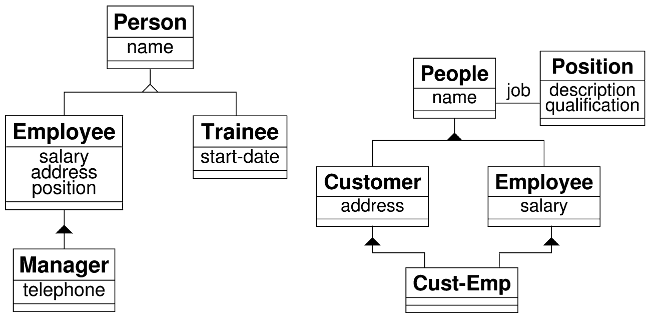


**Fig. 22** Schema 1: company schema and Schema 2: department schema.

**Table 14** Example – global classes expressed by local classes

| Global class | Extension | Local classes |
|---|---|---|
| `Person (C1)` | 1–9 | $\texttt{Person}_1 \cup \texttt{People}_2$ |
| `Trainee (C2)` | 7,8 | $\texttt{Trainee}_1$ |
| `Person-Dep (C4)` | 2–5,7,9 | $\texttt{People}_2$ |
| `Employee (C5)` | 1–6 | $\texttt{Employee}_1$ |
| `Trainee-Cust (C6)` | 7 | $\texttt{Trainee}_1 \cap \texttt{Customer}_2$ |
| `Employee-Dep (C7)` | 2–5 | $\texttt{Employee}_2$ |
| `Manager (C8)` | 1–3 | $\texttt{Manager}_1$ |
| `Manager-Dep (C9)` | 2,3 | $\texttt{Manager}_1 \cap \texttt{Employee}_2$ |
| $\texttt{Position (C10} \cup \texttt{C11)}$ | 10, 11 | $\texttt{Position}_1 \cup \texttt{Position}_2$ |

**Table 15** Example – local classes expressed by global classes

| Local class | Extension | Global classes |
|---|---|---|
| $\texttt{Person}_1$ | 1–8 | $\texttt{Trainee} \cup \texttt{Employee}$ |
| $\texttt{Employee}_1$ | 1–6 | `Employee` |
| $\texttt{Manager}_1$ | 1–3 | `Manager` |
| $\texttt{Trainee}_1$ | 7,8 | `Trainee` |
| $\texttt{People}_2$ | 2–5,7,9 | `Person-Dep` |
| $\texttt{Customer}_2$ | 3,4,7,9 | $\texttt{Person-Dep} \setminus \sigma_{\texttt{customer = false}}\texttt{Employee-Dep}$ |
| $\texttt{Employee}_2$ | 2–5 | `Employee-Dep` |
| $\texttt{Cust-Emp}_2$ | 3,4 | $\sigma_{\texttt{customer = true}}\texttt{Employee-Dep}$ |
| $\texttt{Position}_1$ | 10,11 | `Position` |
| $\texttt{Position}_2$ | 11 | $\sigma_{\text{qualification}\neq\text{null}}\ \texttt{Position}$ |

## 9 Related work

The problem of schema integration in the context of multidatabases exists since database systems have been applied. There is a huge number of publications in this area. Various publications use different terms. For example, for the terms 'multidatabase' and 'federated database system' different definitions exist, e.g. [6,31,45, 52,73]. We followed the definition of a tightly coupled FDBS published in [73]. Our schema architecture is similar to the 5-level-schema-architecture from [73]. In Chapter 2 we compared our architecture with the well known 5-level-schema-architecture.

We did not consider behavior integration. If the behavior is specified in a formal description method, e.g. life cycle diagrams, then results from [53,58] can be used for behavior integration.

For correctness and completeness of schema transformations, mappings between corresponding database instances must be analyzed. Such a mapping analysis together with requirements for the task of schema integration are described in [32,47,48]. Similar to the SIG-formalism proposed in [47,48] we use a small data model and define basic transformation operation. For the correctness proof of every integration step we refer to [62]. Instead, we discussed correctness informally and showed correspondences to SIG-operations. A much more detailed construction of bijective database mappings for the GIM method is given in [62].

Very important for schema integration is the choice of the common data model in which the schema integration is performed. The suitability of different data models as common data model is discussed in [2,33,60]. The favorite data model is usually an object-oriented data model due to its semantical richness. We argue, however, that heterogeneity among object-oriented schemata is often very complex. Furthermore, we distinguish between a data model for homogenization and models for external schemata. As common data model for homogenization we use the Generic Integration Model GIM which enables an efficient algorithm to derive an external schema in a user-friendly data model. The data model GIM was firstly introduced in [64,65].

Schema integration means detecting and resolving schema conflicts. In the literature many different conflict classifications were proposed, e.g. in [1,28,39, 40,50,71,75]. Due to the usage of GIM we have a relatively small number of conflict types which make the schema integration easier than in other data models. Many conflict types of other publications are combinations of our basic conflict types.

A hard problem of schema integration is the detection of conflicts. We have assumed that the designer knows the correspondences. Of course, this is only valid for relatively small schemata. For a deeper discussion about finding correspondences we refer to [24,27,50]. The publications [36,71] use context knowledge and the idea of semantic similarity to detect correspondences.

The structure conflict is a frequently occurring conflict type. [49,50,75], for example, explain the conflict and its resolution. Similar to our approach, the structure conflict is usually resolved by transforming the attribute into a single class.

Attribute conflicts are described in [15,28,39,43]. Following [28,39] they can be sub-classified into conflicts concerning different domains, conflicting integrity constraints, different operations, accuracy, and measures. The designer has to specify a value mapping to relate values from different domains. [75] resolves attribute conflicts by uniting attribute domains and applying integrity constraints for value restrictions. A complex problem is missing injectivity of a mapping which can produce an data loss when a global application inserts a new object and then rereads it again. To the knowledge of the authors no published work solves this problem yet. Our presented approach of splitting attributes transforms this problem into an intensional conflict where one class has more attributes than a corresponding one.

The extensional conflict as one main conflict is subject of many publications, e.g. [4,15,41,46,49,50,72,76]. They usually resolve this conflict directly in an object-oriented model by using specialization. The original classes are often classes of the integrated schema enriched by new super-/subclasses and specialization relationships among them. [18], for example, suggests many operations to resolve a conflict between two classes. Problems arise, however, if two specialization hierarchies with many classes need to be integrated. In this case, the mentioned approaches generate very complex schemata. Furthermore, different variants of conflict resolution are often possible. There are no strict choice rules which help the designer. Therefore, the process of integrating specialization hierarchies is usually very hard for the designer and produces often a huge number of new classes. As mentioned in Sect. 3 most publications do not correctly analyze extensional relations.

[55,79,80] proposed the idea of deriving extensional relations from integrity constraints. Our approach can be extended by this idea to obtain more correct extensional assertions.

During the merging of classes, integrity constraints must be considered. The publications [3,14,20,83,82] point out that integrity constraints can be in a conflict. A conflict occurs when specified extensional relationships are not satisfiable due to different integrity constraints. The publications [3,14,20] handle this conflict as an unsolvable conflict and stop the schema integration. [82,83], however, treat conflicting constraints as subjective and ignore them during the schema integration. [56] resolves the problem of conflicting integrity constraints by weakening them for global classes. This treatment can cause problems when globally inserted objects violate local integrity constraints.

In contrast to the mentioned approaches, the GIM method considers that a conflict as indicating a design failure. Furthermore, the extensional decomposition simplifies the problem, since we have to consider only conflicts between classes with same extensions. Due to the restricted set of constraint types in GIM, conflicting constraints can easily be detected. This avoids problems of undecidability described in [3,14,20,21]. Some ideas of our approach were firstly published in [11-12, 13]. They discuss the relation between constraints and extensional set operations. A more detailed discussion about integrity constraints with respect to schema integration is given in [78].

Objects typically have object identifiers which must be considered during the class merging. [22,64] introduce approaches to tackle the problem of object identifiers in FDBS.

Besides schema integration problems of data integration can occur. It is often very hard to detect SAME objects. Some approaches to this problem are introduced in [9,16,37,38,44,54,84,85,88].

The idea to use mechanisms of the formal concept analysis for the design of object-oriented databases is not new. [87], for example, uses this technique to generate a class hierarchy depending on an intensional analysis. In contrast to our approach, however, [87] does not consider extensional relationships and can, therefore, not directly be used for schema integration. In [63, 65–67] we described how to decompose class extensions for schema integration. This decomposition enables us to use mechanisms of formal concept analysis for schema integration. Our GIM algorithm was firstly published in [67].

In our approach we used a discriminant to avoid loss of data. This problem appears in some publications, e.g. in [42], as conflicting meta information. The resolution of conflicts concerning meta data is topic of [10,59]. [10] introduces a declarative language to overcome meta conflicts. In our approach we use the idea of discriminants published in [25] and adapted it to our GIM scenario.

In the following we discuss publications of different database groups.

### *Publications from navathe and co-authors*

These authors propose an extended entity-relationship model as common data model. Their main focus is on the resolution of extensional conflicts. Depending on binary extensional assertions for resolution they propose to merge, unite, or to intersect classes, or adopt the original classes and establish new specialization relations among them [46]. Besides this conflict, attribute conflicts and conflicting relationships are considered. They resolve these conflicts similar to the extensional conflict by applying the concept of specialization.

Relationships of two ER-diagrams can be conflicting, e.g. with respect to arities, roles, and cardinality numbers. [51] resolves such conflicts very similar to the resolution of the extension conflict.

A very good framework is [43] which describes the resolution of extension conflicts, conflicting relationships, and attribute conflicts. [50] discusses the problem of detecting conflict correspondences between schemata to be integrated.

### *Publications from Saltor, Castellanos and Garcia-Solaco*

The common data model of these publications is the object model BLOOM (Barcelona object-oriented model) introduced in [61]. A motivation using an object model is given in [7,60]. The detection of conflict correspondences is shown in [24,27].

For resolution of extensional conflicts the publications [25,26,59] propose to generalize conflicting classes. In order to avoid data loss they introduce a discriminant attribute. Generalization means, however, not to merge SAME objects to respective global objects. The object merging must be performed in a further step. The use of generalization can produce very complex specialization hierarchies (e.g. in [25, p. 26]), especially if an extensional conflict involves many local classes.

Meta conflicts, which are often called *schematic discrepancies* [42], are resolved by using a discriminant [59].

[28] claim that their integration method does not need to deal with extensional conflicts. They need, however, correspondences between similar classes for which they assume a specific default extensional assertion. In principle, default assertions are also possible in other integration methods. In our opinion, integration without knowing exact extensional relations can produce wrong integrated schemata.

### *Publications from Spaccapietra and Dupont*

These publications consider different conflict classes introduced in [74]. Conflicts are expressed by assertions. Basing on specified assertions, [75] describes different integration rules which resolve these conflicts. This paper considers only identical assertions, i.e. no extensional overlaps or inclusions are regarded. Besides extensional conflicts, conflicting binary relationships are resolved by introducing the concept of paths following references.

Later publications extend the integration method by further extensional conflicts. [18] proposes the integration operations merging, subclass, union, intersection, multi-instantiation, partition, and preservation. A table shows which operation can be applied to which extensional conflict.

[17] points out that binary extensional relationships are not sufficient for a correct extensional analysis. Dupont suggests additional types of non-binary extensional assertions. Binary and the new proposed assertions are, however, still incomplete to exactly model all possible extensional relationships among classes.

*Publications from Ekenberg and Johannesson*

The authors use a logic-based approach to schema transformation and integration. Before performing an integration, the input schemata are normalized, respectively. The common data model has concepts from logic programming and deductive databases. [34] introduces the common data model and some transformation operations. The operations, for example, can be used to transform optional attributes into a specialization and to resolve a structure conflict. [35] argues that due to the translation of the schemata into the common data model they are getting normalized. Therefore, conflicts can be handled relatively easily.

## 10 Conclusion

The contribution of our work is a comprehensive integration method which is based on the theory of formal concept analysis. In contrast to traditional approaches we distinguish between a data model for integration and a data model for schemata of global applications. Due to the integration model GIM the problem of schema integration is transformed into a problem of formal concept analysis resulting in an efficient algorithm to derive global schemata. Furthermore, we focused on the extensional conflict. In contrast to other approaches we are able to resolve complex extensional correspondences. Furthermore, our method allows a semiautomatic derivation of many different external schemata in correspondence to application-specific needs. In order to adapt a schema to specific needs, the designer manipulates the integrated GIM schema accordingly and invokes the derivation algorithm again.

As our simple example has demonstrated, GIM schemata can become very large and unhandy for a designer. This visualization problem, however, is not a real restriction of the GIM method. The GIM representation should be an internal representation within a tool and should be hidden from the designer. The designer controls the integration process by specifying extensional, intensional, and attribute conflicts. The derivation steps can be influenced by using class identifiers and attribute names of the first integrated schema.

Although complex GIM schemata can be hidden, resulting external schemata can become very complex, too. Therefore, we introduced concepts to reduce the complexity. The designer has to find a tradeoff between a design with many null values and a complex external schema without null values. In our opinion, the high complexity of resulting schemata is not a direct result from our GIM approach but a general problem of schema integration. Of course, ignoring extensional relations produces small integrated schemata. But they lack in dealing with redundancy and have many null values. Even worse, the designer does not know where redundancy and null values occur.

At the University of Magdeburg we implemented the main steps of the GIM approach. We successfully used the prototype **SIGMA**$_{\text{Bench}}$in many scenarios, see [70]. Besides schema integration the tool can also be used for view integration, designing specialization hierarchies, and for extending hierarchies by further classes and attributes.

Due to space limitations we were not able to discuss our integration tool **SIGMA**$_{\text{Bench}}$[70], and how database states are explicitly mapped. Furthermore, we explained the attribute conflict very briefly. For more information concerning these issues we refer to [68].

Further research is especially needed for different aspects, e.g.:

– *behavior integration*: The combination of structural and behavior integration requires further research.
– *conflict detection*: Conflict detection is a very time-consuming task for the designer and should therefore be supported by efficient algorithms.
– *schema modifications*: Changing the underlying local schemata requires to adapt the schemata on top of them including their mappings. A new integration process is in general too costly. Therefore, there is a need for an incremental schema integration.

**Acknowledgements** 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").

## References

1. Batini, C., Lenzerini, M.: A methodology for data schema integration in the entity-relationship model. IEEE Transaction on Software Engineering **10**(6), 650–664 (1984)
2. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys **18**(4), 323–364 (1986)
3. Biskup, J., Convent, B.: A formal view integration method. In: Zaniolo, C. (ed.) ACM SIGMOD Record, Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, D.C., vol. 15, pp. 398–407. ACM Press (1986)
4. Bratsberg, S.E.: Evolution and Integration of Classes in Object-Oriented Databases. Dissertation, The Norwegian Institute of Technology, University of Trondheim (1993)
5. Bright, M.W., Hurson, A.R.: Linguistic support for semantic identification and interpretation in multidatabases. In: Kambayashi, Y., Rusinkiewicz, M., Sheth, A. (eds.) Proceedings of the 1st Int. Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan, pp. 306–313. IEEE Computer Society Press (1991)
6. Bukhres, O.A., Elmagarmid, A.K. (eds.): Object-Oriented Multidatabase Systems—A Solution for Advanced Applications. Prentice Hall, Eaglewoods Cliffs, NJ (1996)
7. Castellanos, M., Saltor, F., Garcia-Solaco, M.: A canonical model for the interoperability among object-oriented and relational databases. In: Özsu, M.T., Dayal, U., Valduriez, P. (eds.) Distributed Object Management, pp. 309–314. Morgan Kaufmann Publishers, San Mateo, CA (1994)
8. Catarci, T., Lenzerini, M.: Interschema knowledge in cooperative information systems. In: Huhns, M., Papazoglou, M.P., Schlageter, G. (eds.) Proceedings of the International Conference Intelligent and Cooperating Information Systems, Rotterdam, The Netherlands, pp. 55–62. IEEE Computer Society Press (1993)
9. Chen, A.L.P., Tsai, P.S.M., Koh, J.L.: Identifying object isomerism in multidatabase systems. Distributed and Parallel Databases **4**(2), 143–168 (1996)
10. Chomicki, J., Litwin, W.: Declarative definition of object-oriented multidatabase mappings. In: Özsu, M.T., Dayal, U., Valduriez, P. (eds.) Distributed Object Management, pp. 375–392. Morgan Kaufmann Publishers, San Mateo, CA (1994)
11. Conrad, S., Höding, M., Saake, G., Schmitt, I., Türker, C.: Schema integration with integrity constraints. In: Small, C., Douglas, P., Johnson, R., King, P., Martin, N. (eds.) Advances in Databases, 15th British National Conference on Databases, BNCOD 15, London, UK. Lecture Notes in Computer Science, vol. 1271, pp. 200–214. Springer-Verlag, Berlin (1997)
12. Conrad, S., Schmitt, I., Türker, C.: Dealing with integrity constraints during schema integration. In: Engineering Federated Database Systems EFDBS'97—Proceedings of the International CAiSE'97 Workshop, Barcelona, vol. 6, pp. 13–22. Fakultät für Informatik, Universität Magdeburg (1997)

13. Conrad, S., Schmitt, I., Türker, C.: Considering integrity constraints during federated database design. In: Embury, S.M., Fiddian, N.J., Gray, A.W., Jones, A.C. (eds.) Advances in Databases, 16th British National Conference on Databases, BNCOD 16, Cardiff, Wales. Lecture Notes in Computer Science, vol. 1405, pp. 119–133. Springer-Verlag, Berlin (1998)
14. Convent, B.: Unsolvable problems related to the view integration approach. In: Ausiello, G., Atzeni, P. (eds.) Proceedings of the 1st International Conference Database Theory (ICDT'86), Roma, Italy. Lecture Notes in Computer Science, vol. 243, pp. 141–156. Springer-Verlag, Berlin (1986)
15. Dayal, U., Hwang, H.Y.: View definition and generalization for database integration in a multidatabase system. IEEE Transactions on Software Engineering **10**(6), 628–644 (1984)
16. DeMichiel, L.: Resolving database incompatibility: An approach to performing relational operations over mismatched domains. IEEE Transactions on Knowledge and Data Engineering **1**(4), 485–493 (1989)
17. Dupont, Y.: Resolving fragmentation conflicts in schema integration. In: Loucopoulos, P. (ed.) Entity-Relationship Approach—ER'94, Proceedings of the 13th International Conference on the Entity-Relationship Approach, Manchester, UK. Lecture Notes in Computer Science, vol. 881, pp. 513–532. Springer-Verlag, Berlin (1994)
18. Dupont, Y., Spaccapietra, S.: Schema integration engineering in cooperative databases systems. In: Yetongnon, K., Hariri, S. (eds.) Proceedings of the 9th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS'96), Dijon, France, September 1996, pp. 759–765. International Society for Computers and Their Application, Six Forks Road, Releigh, NC (1996)
19. Duquenne, V.: Contextual implications between attributes and some properties of finite lattices. In: Ganter, B., Wille, R. (eds.) Beiträge zur Begriffsanalyse, Chap. 10, pp. 213–239. B. I.-Wissenschaftsverlag, Mannheim (1987)
20. Ekenberg, L., Johannesson, P.: Conflictfreeness as a basis for schema integration. In: Bhalla, S. (ed.) Information Systems and Data Management, Proceedings of the 6th Conference, CISMOD'95, Bombay, India. Lecture Notes in Computer Science, vol. 1006, pp. 1–13. Springer-Verlag, Berlin (1995)
21. Ekenberg, L., Johannesson, P.: A formal basis for dynamic schema integration. In: Thalheim, B. (ed.) Conceptual Modelling—ER'96, Proceedings of the 15th International Conference, Cottbus, Germany. Lecture Notes in Computer Science, vol. 1157, pp. 211–226. Springer-Verlag, Berlin (1996)
22. Eliassen, F., Karlsen, R.: Interoperability and object identity. ACM SIGMOD Record **20**(4), 25–29 (1991)
23. Ganter, B., Wille, R.: Formal Concept Analysis. Springer-Verlag, Berlin/Heidelberg (1998)
24. Garcia-Solaco, M., Castellanos, M., Saltor, F.: Discovering interdatabase resemblance of classes for interoperable databases. In: Schek, H.J., Sheth, A.P., Czejdo, B.D. (eds.) Proceedings of the 3rd International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems (RIDE-IMS'93), Vienna, Austria, pp. 26–33. IEEE Computer Society Press (1993)
25. Garcia-Solaco, M., Castellanos, M., Saltor, F.: A Semantic-discriminated approach to integration in federated databases. In: Laufmann, S., Spaccapietra, S., Yokoi, T. (eds.) Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'95). Vienna, Austria, pp. 19–31 (1995)
26. Garcia-Solaco, M., Saltor, F.: Discriminated Operations for Interoperable Databases. In: International Workshop on Interoperability in Multidatabase Systems, Kyoto (1991)
27. Garcia-Solaco, M., Saltor, F., Castellanos, M.: A structure based schema integration methodology. In: Yu, P.S., Chen, A.L.P. (eds.) Proceedings of the 11th IEEE International Conference on Data Engineering, ICDE'95, Taipei, Taiwan, pp. 505–512. IEEE Computer Society Press, Los Alamitos, CA (1995)
28. Garcia-Solaco, M., Saltor, F., Castellanos, M.: Semantic heterogeneity in multidatabase systems. In: Bukhres, O.A., Elmagarmid, A.K. (eds.) Object-Oriented Multidatabase Systems—A Solution for Advanced Applications, chap. 5, pp. 129–202. Prentice Hall, Eaglewoods Cliffs, NJ (1996)
29. Gertz, M., Schmitt, I.: Data integration techniques based on data quality aspects. In: Schmitt, I., Türker, C., Hildebrandt, E., Höding, M. (eds.) Proceedings 3. Workshop "Föderierte Datenbanken", Magdeburg, pp. 1–19. Shaker Verlag, Aachen (1998)

30. Gottlob, G., Schrefl, M., Röck, B.: Extending object-oriented systems with roles. ACM Transactions on Information Systems **14**(3), 268–296 (1996)
31. Heimbigner, D., McLeod, D.: A federated architecture for information management. ACM Transactions on Office Information Systems **3**(3), 253–278 (1985)
32. Hull, R.: Relative information capacity of simple relational database schemata. SIAM Journal on Computing **15**(3), 856–886 (1986)
33. Hurson, A.R., Bright, M.W.: Object-oriented multidatabase systems. In: Bukhres, O.A., Elmagarmid, A.K. (eds.) Object-Oriented Multidatabase Systems—A Solution for Advanced Applications, chap. 1, pp. 1–36. Prentice Hall, Eaglewoods Cliffs, NJ (1996)
34. Johannesson, P.: Schema transformations as an aid in view integration. In: Rolland, C., Bodart, F., Cauvet, C. (eds.) Proceedings of the 5th Conference on Advanced Information System Engineering (CAiSE'93), Paris, France. Lecture Notes in Computer Science, vol. 685, pp. 71–92. Springer-Verlag, Berlin (1993)
35. Johannesson, P.: Using conceptual graph theory to support schema integration. In: Elmasri, R.A., Kourajian, V., Thalheim, B. (eds.) Entity-Relationship Approach—ER'93, Proceedings of the 12th International Conference on the Entity-Relationship Approach, Arlington, Texas, December, 1993. Lecture Notes in Computer Science, vol. 823, pp. 283–296. Springer-Verlag (1994)
36. Kashyap, V., Sheth, A.: Semantic and schematic similarities between database objects: A context-based approach. The VLDB Journal **5**(4), 276–304 (1996)
37. Kent, W.: A rigorous model of object reference, identity, and existence. Journal of Object-Oriented Programming pp.28–36 (1991)
38. Kent, W., Ahmed, R., Albert, J., Ketabchi, M., Shan, M.C.: Object identification in multidatabase systems. In: Hsiao, D.K., Neuhold, E.J., Sacks-Davis, R. (eds.) Proceedings of the IFIP WG 2.6 Working Conference on Interoperable Database Systems (DS-5), Victoria, Australia, November, 1992, pp. 313–330. North-Holland, Amsterdam (1993)
39. Kim, W., Choi, I., Gala, S., Scheevel, M.: On Resolving Schematic Heterogeneity in Multidatabase Systems. In: W. Kim (ed.) Modern Database Systems, pp. 521–550. ACM Press, New York, NJ (1995)
40. Kim, W., Seo, J.: Classifying Schematic and Data Heterogeneity in Multidatabase Systems. IEEE Computer **24**(12), 12–18 (1991)
41. Klas, W., Schrefl, M.: Meta Classes and Their Applications—Data Model Tailoring and Database Integration. Lecture Notes in Computer Science, vol. 943. Springer-Verlag, Berlin (1995)
42. Krishmamurthy, R., Litwin, W., Kent, W.: Interoperability of heterogeneous databases with schematic discrepancies. In: Kambayashi, Y., Rusinkiewicz, M., Sheth, A. (eds.) Proceedings of the 1st International Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan, pp. 144–151. IEEE Computer Society Press (1991)
43. Larson, J.A., Navathe, S.B., Elmasri, R.: A theory of attribute equivalence in databases with application to schema integration. IEEE Transactions on Software Engineering **15**(4), 449–463 (1989)
44. Lim, E.P., Srivastava, J., Prabhakar, S., Richardson, J.: Entity identification in database integration. In: Elmagarmid, A., Neuhold, E. (eds.) Proceedings of the 9th IEEE International Conference on Data Engineering, ICDE'93, Vienna, Austria, pp. 294–301. IEEE Computer Society Press, Los Alamitos, CA (1993)
45. Litwin, W., Mark, L., Roussopoulos, N.: Interoperability of multiple autonomous databases. ACM Computing Surveys **22**(3), 267–293 (1990)
46. Mannino, M.V., Navathe, B.N., Effelsberg, W.: A rule-based approach for merging generalization hierarchies. Information Systems **13**(3), 257–272 (1988)
47. Miller, R.J., Ioannidis, Y.E., Ramakrishnan: The use of information capacity in schema integration and translation. In: Baker, S., Agrawal, R., Bell, D. (eds.) Proceedings of the 19th International Conference on Very Large Data Bases (VLDB'93), Dublin, Ireland, pp. 120–133. Morgan Kaufmann Publishers, San Francisco, CA (1993)
48. Miller, R.J., Ioannidis, Y.E., Ramakrishnan, R.: Schema equivalence in heterogeneous systems: Bridging theory and practice. Information Systems **19**(1), 3–31 (1994)
49. Motro, A.: Superviews: Virtual integration of multiple databases. IEEE Transactions on Software Engineering **13**(7), 785–798 (1987)

50. Navathe, S., Savasere, A.: A schema integration facility using object-oriented data model. In: Bukhres, O.A., Elmagarmid, A.K. (eds.) Object-Oriented Multidatabase Systems—A Solution for Advanced Applications, chap. 4, pp. 105–128. Prentice Hall, Eaglewoods Cliffs, NJ (1996)
51. Navathe, S.B., Elmasri, R., Larson, J.A.: Integrating user views in database design. IEEE Computer **19**(1), 50–62 (1986)
52. Pitoura, E., Bukhres, O., Elmagarmid, A.K.: Object orientation in multidatabase systems. ACM Computing Surveys **27**(2), 141–195 (1995)
53. Preuner, G.: Definition of Behavior in Object-Oriented Databases by View Integration. Dissertationen zu Datenbanken und Informationssystemen, vol. 53. Infix-Verlag, Sankt Augustin (1999)
54. Pu, C.: Key equivalence in heterogenous databases. In: Kambayashi, Y., Rusinkiewicz, M., Sheth, A. (eds.) Proceedings of the 1st International Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan, pp. 314–316. IEEE Computer Society Press (1991)
55. Ramesh, V., Ram, S.: Integrity constraint integration in heterogeneous databases: An enhanced methodology for schema integration. Information Systems **22**(8), 423–446 (1997)
56. Reddy, M.P., Prasad, B.E., Gupta, A.: Formulating global integrity constraints during derivation of global schema. Data & Knowledge Engineering **16**(3), 241–268 (1995)
57. Reddy, M.P., Prasad, B.E., Reddy, P.G., Gupta, A.: A methodology for integration of heterogeneous databases. IEEE Transactions on Knowledge and Data Engineering **6**(6), 920–933 (1994)
58. Saake, G., Hartel, P., Jungclaus, R., Wieringa, R., Feenstra, R.: Inheritance conditions for object life cycle diagrams. In: Lipeck, U., Vossen, G. (eds.) Workshop Formale Grundlagen für den Entwurf von Informationssystemen, Tutzing, 3/94, pp. 79–89. Universität Hannover (1994)
59. Saltor, F., Castellanos, Garcia-Solaco, M.: Overcoming schematic discreprancies in interoperable databases. In: Hsiao, D.K., Neuhold, E.J., Sacks-Davis, R. (eds.) Interoperable Database Systems (DS-5), Proceedings of the IFIP WG 2.6 Database Semantics Conference, Lorne, Victoria, Australia, November, 1992, pp. 191–205. North-Holland, Amsterdam (1993)
60. Saltor, F., Castellanos, M., Garcia-Solaco, M.: Suitability of data models as canonical models for federated databases. ACM SIGMOD Record **20**(4), 44–48 (1991)
61. Saltor, F., Castellanos, M., Garcia-Solaco, M., Kudrass, T.: Modelling Specialization as BLOOM semilattices. In: Kangassalo, H., Jaakkola, H., Ohsuga, S., Wangler, B. (eds.) Information Modelling and Knowledge Bases VI, Japan, pp. 447–467. IOS Press, Amsterdam (1995)
62. Schmitt, I.: Schema Integration for the Design of Federated Databases. Dissertationen zu Datenbanken und Informationssystemen, vol. 43. Infix-Verlag, Sankt Augustin (1998). (In German)
63. Schmitt, I., Conrad, S.: Restructuring class hierarchies for schema integration. In: Topor, R., Tanaka, K. (eds.) Database Systems for Advanced Applications '97, Proceedings of the 5th International Conference, DASFAA'97, Melbourne, Australia, pp. 411–420. World Scientific Publishing, Singapore (1997)
64. Schmitt, I., Saake, G.: Managing object identity in federated database systems. In: Papazoglou, M. (ed.) OOER'95: Object-Oriented and Entity-Relationship Modeling, Proceedings of the 14th International Conference, Gold Coast, Australia. Lecture Notes in Computer Science, vol. 1021, pp. 400–411. Springer-Verlag, Berlin (1995)
65. Schmitt, I., Saake, G.: Integration of inheritance trees as part of view generation for database federations. In: Thalheim, B. (ed.) Conceptual Modelling—ER'96, Proceedings of the 15th International Conference, Cottbus, Germany. Lecture Notes in Computer Science, vol. 1157, pp. 195–210. Springer-Verlag, Berlin (1996)
66. Schmitt, I., Saake, G.: Schema integration and view generation by resolving intensional and extensional overlappings. In: Yetongnon, K., Hariri, S. (eds.) Proceedings of the 9th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS'96), Dijon, France, September 1996, pp. 751–758. International Society for Computers and Their Application, Six Forks Road, Releigh, NC (1996)

67. Schmitt, I., Saake, G.: Merging inheritance hierarchies for database integration. In: Halper, M. (ed.) Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, CoopIS'980. New York, USA, pp. 322–331. IEEE Computer Society Press, Los Alamitos, CA (1998)
68. Schmitt, I., Saake, G.: Integrating database schemata using the GIM method. Preprint 20, Fakultät für Informatik, Universität Magdeburg (1999)
69. Schmitt, I., Türker, C.: Refining extensional relationships and existence requirements for incremental schema integration. In: Gardarin, G., French, J., Pissinou, N., Makki, K., Bougamin, L. (eds.) Proceedings of the 7th ACM CIKM International Conference on Information and Knowledge Management. Bethesda, Maryland, USA, pp. 322–330. ACM Press, New York (1998)
70. Schwarz, K., Schmitt, I., Türker, C., Höding, M., Hildebrandt, E., Balko, S., Conrad, S., Saake, G.: Design support for database federations. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., Métais, E. (eds.) Conceptual Modeling—ER'99, 18th International Conference on Conceptual Modeling, Paris, France, November 15–18, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1728, pp. 445–459. Springer-Verlag, Berlin (1999)
71. Sheth, A., Kashyap, V.: So far (schematically) yet so near (semantically). In: Hsiao, D.K., Neuhold, E.J., Sacks-Davis, R. (eds.) Interoperable Database Systems (DS-5), Proceedings of the IFIP WG 2.6 Database Semantics Conference, Lorne, Victoria, Australia, November, 1992, pp. 283–312. North-Holland, Amsterdam (1993)
72. Sheth, A.P., Gala, S.K., Navathe, S.B.: On automatic reasoning for schema integration. International Journal of Intelligent and Cooperative Information Systems **2**(1), 23–50 (1993)
73. Sheth, A.P., Larson, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys **22**(3), 183–236 (1990)
74. Spaccapietra, S., Parent, C.: Conflicts and correspondence assertions in interoperable databases. ACM SIGMOD Record **20**(4), 49–54 (1991)
75. Spaccapietra, S., Parent, C., Dupont, Y.: Model independent assertions for integration of heterogeneous schemas. The VLDB Journal **1**(1), 81–126 (1992)
76. Thieme, C., Siebes, A.: Schema integration in object-oriented databases. In: Rolland, C., Bodart, F., Cauvet, C. (eds.) Proceedings of the 5th Conference on Advanced Information System Engineering (CAiSE'93), Paris, France. Lecture Notes in Computer Science, vol. 685, pp. 55–70. Springer-Verlag, Berlin (1993)
77. Tsichritzis, D.C., Klug, A.: The ANSI/X3/SPARC DBMS framework report of the study group on database management systems. Information Systems **3**(3), 173–191 (1978)
78. Türker, C.: Semantic Integrity Constraints in Federated Database Schemata. Dissertation, University of Magdeburg, Germany (1999)
79. Türker, C., Saake, G.: Deriving relationships between integrity constraints for schema comparison. In: Litwin, W., Morzy, T., Vossen, G. (eds.) Advances in Databases and Information Systems, Proceedings Second East-European Symposium, ADBIS'98, Poznań, Poland. Lecture Notes in Computer Science, vol. 1475, pp. 188–199. Springer-Verlag, Berlin (1998)
80. Türker, C., Saake, G.: Consistent handling of integrity constraints and extensional assertions for schema integration. In: Eder, J., Rozman, I., Welzer, T. (eds.) Advances in Databases and Information Systems, Proceedings Third East-European Symposium, ADBIS'99, Maribor, Slovenia. Lecture Notes in Computer Science, vol. 1691, pp. 31–45. Springer-Verlag, Berlin (1999)
81. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, Volume II: The New Technologies. Computer Science Press, Rockville, MD (1989)
82. Vermeer, M.W.W.: Semantic Interoperability for Legacy Databases. 97-11. Centre for Telematics and Information Technology, Enschede, The Netherlands (1997)
83. Vermeer, M.W.W., Apers, P.M.G.: The role of integrity constraints in database interoperation. In: Vijayaraman, T.M., Buchmann, A.P., Mohan, C., Sarda, N.L. (eds.) Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96), Bombay, India, pp. 425–435. Morgan Kaufmann Publishers, San Francisco, CA (1996)
84. Wang, K., Zhang, W.: Detecting data inconsistency for multidatabases. In: Yetongnon, K., Hariri, S. (eds.) Proceedings of the 9th ISCA International Conference on Parallel and Distributed Computing Systems (PDCS'96), Dijon, France, pp. 657–663. International Society for Computers and Their Application, Six Forks Road, Releigh, NC (1996)

85. Wang, Y., Madnick, S.: The Interdatabase Instance Identification Problem in Integrating Autonomous Systems. In: Proceedings of the 5th IEEE International Conference on Data Engineering, Los Angeles, pp. 46–55 (1989)
86. Wieringa, R.J., De Jonge, W., Spruit, P.A.: Using dynamic classes and role classes to model object migration. Theory and Practice of Object Systems **1**(1), 61–83 (1995)
87. Yahia, A., Lakhal, L., Cicchetti, R., Bordat, J.P.: i$O_2$: An algorithmic method for building inheritance graphs in object database design. In: Thalheim, B. (ed.) Conceptual Modelling—ER'96, Proceedings of the 15th International Conference, Cottbus, Germany. Lecture Notes in Computer Science, vol. 1157, pp. 422–437. Springer-Verlag, Berlin (1996)
88. Zhou, G., Hull, R., King, R., Franchitti, J.C.: Using object matching and materialization to integrate heterogeneous databases. In: Laufmann, S., Spaccapietra, S., Yokoi, T. (eds.) Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS'95). 1995, Vienna, Austria, pp. 4–18 (1995)