Integrating Database Schemata using the GIM Method[1]

Ingo Schmitt Gunter Saake

Otto-von-Guericke-Universität Magdeburg
Institut für Technische Informationssysteme
Postfach 4120, D–39016 Magdeburg, Germany
E-mail: {schmitt|saake}@iti.cs.uni-magdeburg.de

September 1999

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

Abstract

Integrating heterogeneous schemata is a main task in the scenario of federated and heterogeneous database design. However, most proposed methodologies for schema integration suffer from complexity and insufficient handling of extensional relations. They mostly use an object-oriented data model both for the process of schema integration and for the integrated schema. The GIM approach, however, distinguishes between these aspects. The process of schema integration is performed using the Generic Integration Model designed for the integration task. Using the theory of formal context analysis an integrated schema in an other data model can be derived semi-automatically. The resulting schema is optimized with respect to minimality and understandability. The GIM approach gives a methodology for schema integration consisting of different steps. After translating original schemata into the Generic Integration Model they are compared and homogenized. Extensional relations are playing a important role for homogenization. When all schema conflicts are resolved they are merged into one GIM schema. Applying mechanisms of formal context analysis provides object-oriented, integrated schemata. The design steps are partially supported by algorithms to reduce the complexity of the process.

Contents

Chapter 1

Introduction

Database schema integration is a topic of database engineering since the early days of database management systems. It is needed in several scenarios: view integration during database design, integration of databases in data warehouses, building of federated databases, and schema extension during schema evolution.

In this paper we concentrate on schema integration as part of building federated databases. Federated databases establish a virtual global view on existing databases while keeping the participating systems partially autonomous. Federated database systems FDBS are discussed since the eighties and are recently becoming popular due to the problem of integrating worldwide databases in the Internet.

From the view of schema integration, FDBS are the most complex case. Besides schema information, the actual databases as their instances are already existing and have to be preserved. Moreover, we may have to handle the intrinsic heterogeneity of databases developed in different companies and organizations. Since the integration is virtual, we have to define explicit mappings processed by the FDBMS. Last but not least, some scenarios even require global updates which requires the inverse mapping of database states and updates.

Besides the homogenization needed to overcome the heterogeneity, we have in FDBS the problem of *redundancy* arising from independently developed databases. Parts of the information are stored redundantly in several databases. This redundancy has to be managed in the integration process.

We present the *GIM method* as a new approach to schema integration. GIM stands for *Generic Integration Model*. Some selected algorithms being used in the GIM method are already presented in [SS96a, SS96b, SS98]. However, this paper is the first presentation of the whole method covering the complete integration process. In contrast to other methods, GIM is a *semantically poor* database model based on a simple object model. GIM does not support the variety of modeling concepts which can be found for example in semantic data models or object models. Thus, a GIM presentation can be used as an intermediate design step only. However, the lack of semantic modeling concepts allows the use of schema transformation algorithms for design tasks which have to be performed

manually in other approaches.

The GIM transformation process covers all phases of schema integration needed in FDBS design. The single steps are formalized and can therefore be supported by design tools which produce process-able transformation rules for data conversion, queries and updates. Since GIM is not adequate as an application model, an object-oriented database schema can be generated as application database model.

For the following presentation, we assume to integrate two existing databases. For n-ary integration we refer to the discussion in [BLN86]. We concentrate on structural (i.e., static) aspects and will not discuss dynamic aspects of database integration like behavior integration [Pre99].

Before starting the technical part of the presentation, we want to fix some notational conventions.

- Redundancy between two databases is expressed by establishing a SAME *relation* between database objects. The SAME relation has of course to be an equivalence relation. Figure 1.1 depicts the underlying assumption that different database objects model the same real world entity.

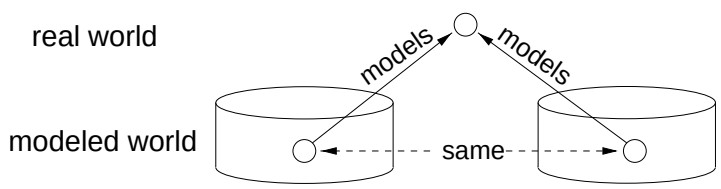


Figure 1.1: SAME relation

- We group objects into classes. The *intension* or the type of a class is defined by a set of typed attributes.

- The *extension* of a class c, denoted as $\mathcal{E}xt_c$, is defined as the set of instances of c at a certain instant of time.

- The *extensional analysis* is the non-trivial process of inspecting class extensions, application programs and developer knowledge to detect extensional relationships between different classes which are independently valid from specific database states, i.e. specific instants of time. Those extensional relationships are specified by extensional *assertions* like inclusion, disjointness, etc.

  The classical approach to extensional analysis is to state binary extensional assertion, for example stating that the classes `Man` and `Woman` are disjoint. We will later on discuss the deficiencies of using binary relationships only.

This paper is organized as follows. Section 2 discusses the overall requirements of schema integration and the quality criteria for design steps and resulting database 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 defines the GIM data model used within the integration process. The translation of the input schemata into the common data model GIM is described in Section 5. The GIM schemata are then compared in order to find conflict correspondences. Section 6 shows the conflict resolving by transforming schemata. After this step the resulting two homogeneous GIM schema are merged into the integrated schema following the design step described in Section 7. To obtain an integrated schema in a user-friendly data model Section 8 discusses algorithms to derive external schemata from the integrated GIM schema. An external schema can be adapted to a certain application view. The design process is supported by the design tool **SIGMA**$_{Bench}$ introduced in Section 9. After discussing related work in Section 10 we conclude and summarize our work in Section 11. Some screen shots from our design tool are presented in the appendix.

Chapter 2

Problem Description and Requirements

For the following discussions we assume the following design scenario. The starting point of the integration are two schemata of existing databases. Additionally, the designer has knowledge from other sources about the database semantics which can be used in the integration process.

Aim of the integration process is the construction of *one integrated schema* together with several *external views* on the integrated databases. Besides these constructed schemata we need processable mapping descriptions to enable queries and updates on these schemata. Figure 2.1 visualizes this design scenario.

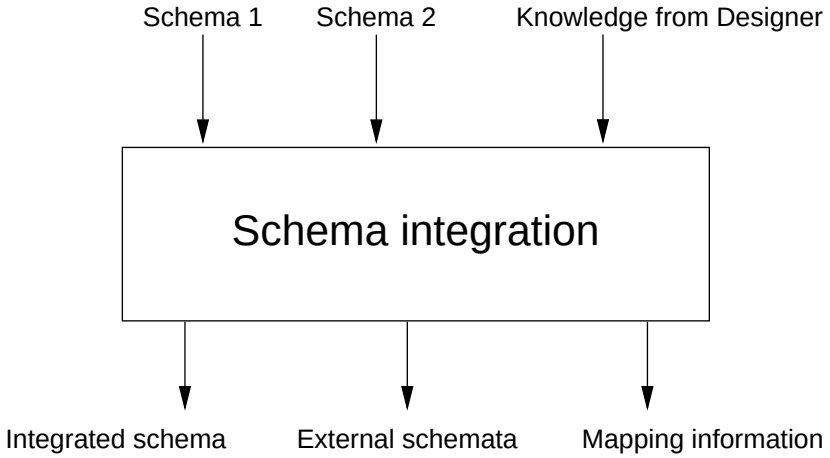


Figure 2.1: Input and output of the schema integration

Some design approaches discuss the construction of the integrated schema only and do not consider external views. Besides the benefits of data independence resulting from the well-known three-level schema architecture consisting of internal, conceptual and external level, we have some aspects more specific to our scenario:

- The *migration* of applications from local databases towards the global federation is supported by external schemata resembling local views.
- Multiple external schemata allow views on the integrated databases using different modeling concepts at the same time, for example object schemata with or without multiple inheritance or pure relational views.
- Sometimes the required application data model for the integrated databases is not suitable for automatic integration algorithms. In this case, an intermediate representation of the integrated database can be used.

These aspects can be supported by a method allowing semi-automatic generation of external views.

2.1 General Requirements

This design scenario implies some intrinsic problems, which have to be handled by a suitable design method:

- The two input database schemata may be specified using *heterogeneous data models*. This data model heterogeneity can be solved by translating them into a *common data model* used for integration purposes.
- The translated schemata may contain *redundant schema parts* because of non-normalized and non-integrated databases. We call a schema part *redundant* if it describes redundantly stored database parts. This redundancy has to be detected and resolved.
- The problem of *schema heterogeneity* arises because the same semantic concepts maybe modeled differently in different database schemata.
- The necessary designer knowledge is mostly neither explicit, exact, consistent, nor complete. This makes it hard to use it in automatic design steps.

These problems lead to (partly formalized) requirements on the result of the integration process. One early list of requirements was published in [BLN86]. In this paper, an integrated schema has to fulfill the following requirements:

- Completeness,
- Correctness,
- Minimality,
- Understandability.

These requirements will be discussed in the following sections in some detail and even partly be formalized. Before doing this, we have to detail the transformation process and to identify some intermediate schema representations.

2.2 Involved Schema Levels

As a first step we analyze the different schema levels occurring during the design process. We start with two local schemata S_1 and S_2. The result of the integration process is one integrated schema S^I and possibly several external schemata S_i^E.

We propose several intermediate representations to ensure the stated requirements for the integration process and result:

1. The two schemata S_1 and S_2 are translated into the *common data model* used for the integration. This step resolves data model heterogeneity and produces the schemata S_1^C and S_2^C.
2. The schemata S_1^C and S_2^C are *homogenized* resulting in two schemata S_1^H and S_2^H to overcome schema heterogeneity.
3. The homogenized schemata S_1^H and S_2^H are *merged* into the integrated schema S^I.
4. Now several external views with schemata $S_1^E, \ldots S_i^E$ are derived. For simplicity, we will consider only one view with schema S^E representing an understandable view on the integrated database.

Table 2.1 summarizes the requirements concerning understandability and minimality for these schemata. There is of course a trade-off between understandability and minimality.

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

Table 2.1: Requirements for Schemata

For example, a universal relation would be minimal but not really understandable.

2.3 Completeness and Correctness

For discussing completeness and correctness as design requirements, we have to analyze the *mappings* between the discussed schema levels. These mappings have to be extended to *transform databases states*, too, to ensure the desired properties.

We denote the set of databases states for a given schema S with $I(S)$ (notated with the letter I for *instantiation*). We relate actual database states of the variant schema levels with the following functions φ:

1. The homogenization into the common data model results in the following function:

$$\begin{aligned} \varphi_1^C &\subseteq I(S_1) \rightarrow I(S_1^C) \\ \varphi_2^C &\subseteq I(S_2) \rightarrow I(S_2^C) \end{aligned}$$

2. For the homogenized schemata we have the following transformations:

$$\begin{aligned} \varphi_1^H &\subseteq I(S_1^C) \rightarrow I(S_1^H) \\ \varphi_2^H &\subseteq I(S_2^C) \rightarrow I(S_2^H) \end{aligned}$$

3. The merging phase results in a function φ^I:

$$\varphi^I \subseteq (I(S_1^H) \times I(S_2^H)) \rightarrow I(S^I)$$

   It should be noted that φ^I has to be defined for states only which are in the time relation $SameTime(I(S_1^H), I(S_2^H))$. This is a consequence of the fact that extensional relationships, e.g. inclusion dependencies between classes of different databases, are state-dependent, i.e., are defined for states at same instant of time.

4. The derivation of an external schema is supported by the following function:

$$\varphi^E \subseteq I(S_I) \rightarrow I(S_E)$$

We define the function φ as concatenation of the different functions. Again, these functions are defined for states being in time relation only. Thus, for all $i_1 \in I(S_1)$ and $i_2 \in I(S_2)$ the following derived functions are defined (for all functions $\varphi_i^C, \varphi_i^H, \varphi^I, \varphi^E$):

$$SameTime(i_1, i_2) \Rightarrow \varphi(i_1, i_2) \stackrel{Def}{=} \varphi^E(\varphi^I(\varphi_1^H(\varphi_1^C(i_1)), \varphi_2^H(\varphi_2^C(i_2))))$$

For these functions we state some requirements which have to be satisfied. For each function φ^x between a source schema S and a transformed schema S' we require:

1. The function (and its inverse mapping) has to be effectively computable.

2. The function has to be complete:

$$\forall i \in I(S) : \exists i' \in I(S') : \varphi(i) = i'$$

3. The functions are surjective:

$$\forall i' \in I(S') : \exists i \in I(S) : \varphi(i) = i'$$

   This property is relevant because we want to allow updates on the integrated database and need therefore an inverse mapping.

4. For some functions, we require injectivity, too:

$$\forall i_1, i_2 \in I(S) : \varphi(i_1) = \varphi(i_2) \Rightarrow i_1 = i_2$$

   Injectivity avoids loss of information during the integration process. This is required for the functions φ^C and φ^H, only. Because we want to remove data inconsistencies during the merging, φ^I has not necessarily to be injective. For the construction of external views we allow selection of relevant database parts. Therefore, φ^E is in general not injective, too.

In contrast to requirement 4, the requirements 1 to 3 have to be satisfied by all functions. The functions φ^C and φ^H are the only bijective mappings, which preserve the information capacity following [Hul86].

In contrast to other authors [MIR93], we do *not* require injectivity for the merging process. However, to allow global updates, we require surjectivity. As a consequence, we need *global integrity constraints* to restrict possible global databases states appropriately.

The high complexity of constructing the required functions, especially the removal of conflicts in the homogenization step, requires the support by a formalized design method.

2.4 Requirements for the Integration Process

To support the design process of a database federation, we surely need supporting tools. Therefore, we have to develop algorithms for solving specific problems in constructing intermediate schemata and related mappings. For the design method in general we have the usual requirements:

- The method should be *complete*, i.e. should handle all classes of conflicts and should construct all discussed schema levels (including the external!).
- It should be *correct* w.r.t. the discussed requirements for schema and database mappings.
- The effort of manual intervention should be *minimal*, i.e. the degree of automation should be as high as possible.
- Both the design steps and resulting schemata should be *understandable* by designers.

2.5 Relation to the Five-Level Schema Architecture

The 5-level schema architecture of Sheth and Larson [SL90] is usually accepted as a reference model for comparing federation approaches. Figure 2.2 relates our schema levels to those of [SL90].

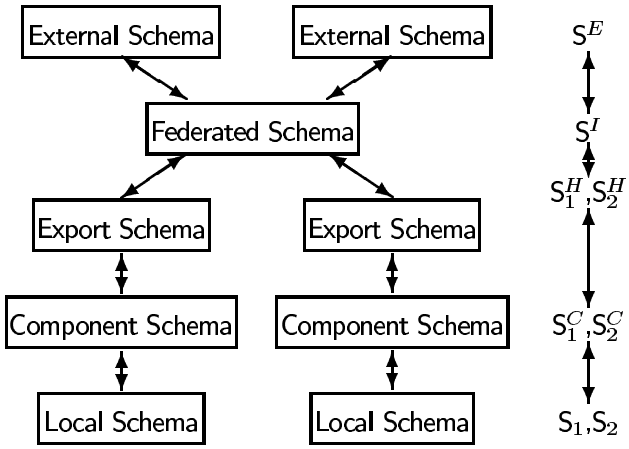


Figure 2.2: Schema architecture

As a difference, we do not consider explicit export schemata but homogenized schemata S^H. Since the GIM approach does not use the integrated schema as an application schema of the federation, we have renamed it from federated schema to integrated schema.

2.6 Deficiencies of Existing Methods

Schema integration is the bottleneck of building database federations [NS96]. One reason can be found in common deficiencies of most existing approaches, which can be listed as follows:

- Most approaches consider only specific steps of the integration process and do not present a method covering all steps from heterogeneous input databases up to the external schema derivation.
- Approaches often do not specify the corresponding database mappings φ.
- Another weak point are missing algorithms supporting design steps.
- One weak point, which we want especially to address, is the missing consideration of extensional conflicts. Extensional conflicts express for which classes same objects can occur. To meet the requirements for minimality and correctness we have to know extensional class overlaps. To the authors knowledge no approach analyzes extensional relations completely. Most approaches enable the specification of binary extensional assertions comparing the extensions of two classes only. As shown in [SS96b] binary assertion are not powerful enough to express all extensional relations among classes. As result of an incomplete specification some same objects are not detected and therefore not merged into global, respective objects or some global class extensions are always empty.

Basing on extensional assertions most approaches offer various operations to resolve this conflict. Usually, for one certain assertion more then one resolution operation can be applied, see for example [DS96]. The integration designer has to choose the suitable operation. There are no rules for the designer to select the best operation. Furthermore, the operations often resolve a conflict between two classes only. If extensional conflicts involving many classes are broken down to binary conflicts and the binary operations are applied to them the designer often obtains a very complex schema with many unnecessary classes.

Some approaches, e.g. [GSC96, RPRG94, RPG95], do not analyze extensional relations. They merge semantically relating classes into one global class. The set of attributes of the global class is computed by the union of the local class attributes. Without extensional analysis, however, the designer does not know for which potential global object and for which attribute a value exists. Therefore, many null values occur and violate the requirement for a sound design.

Our approach aims to overcome these problems. However, due to the complexity of the problem, there is no hope for a complete solution in a single shot. Our focus in this paper is on the following aspects:

- A complete and correct handling of the extensional analysis (presented in Section 6.2).
- Adaption of algorithms of formal context analysis for schema analysis and schema derivation (Section 8).

  This adaption results in relatively simple algorithms for deriving minimal integrated schemata and object-oriented views.
- Specifying database mappings φ.
- Derivation of external schemata.

Chapter 3

Example

For presenting the steps of the GIM method, we have chosen a small integration scenario showing most of the integration problems in a relatively small application.

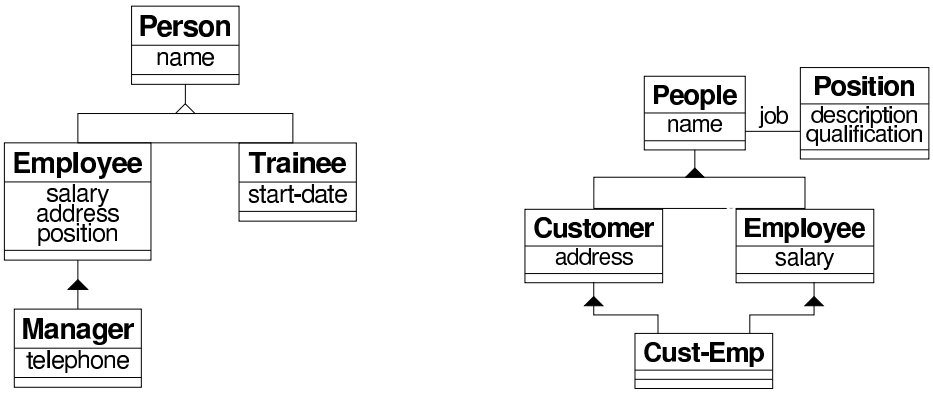


Figure 3.1: Schema$_1$: company schema and schema$_2$: department schema

Figure 3.1 shows two database schemata in an OMT-like notation. The first database stores information about people working in a company. Those persons are partitioned into employees and trainees; managers are a subset of employees. The second database contains data about persons relevant for a department of this company. We will distinguish classes from both databases with subscripts 1 and 2.

Table 3.1 lists some constraints for both databases, which are relevant for the integration process.

Local Class	Integrity Constraint
`Manager`$_1$	`salary` ≥ 3500
`Employee`$_2$	`salary` ≥ 3000

Table 3.1: Integrity Constraints

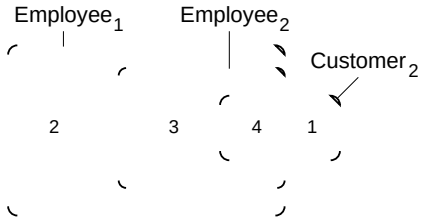
The GIM method especially attacks the problem of extensional relationships (sometimes called 'extensional conflicts'). In our small example, we identify the following extensional relations as conditions which are relevant for the integration process:

1. The *specialization* constructor leads to subset relations between the instances of subclass and superclass.
2. There is a partition of $\texttt{Person}_1$ into $\texttt{Employee}_1$ and $\texttt{Trainee}_1$. The extensions of $\texttt{Employee}_1$ and $\texttt{Trainee}_1$ are disjoint. Their union is equal to the extension of $\texttt{Person}_1$.
3. Each person in $\texttt{People}_2$ is also member of $\texttt{Customer}_2$ or $\texttt{Employee}_2$ or of both.
4. Each person in the intersection of $\texttt{Customer}_2$ and $\texttt{Employee}_2$ is included in $\texttt{Cust-Emp}_2$.
5. The instances of $\texttt{Employee}_2$ build at each instant of time a subset of those of $\texttt{Employee}_1$. In other words, the class $\texttt{Employee}_1$ constitutes the set of *all* employees of the company (including all departments).
6. Persons in the intersection of $\texttt{Employee}_1$ and $\texttt{Customer}_2$ are included in $\texttt{Employee}_2$ — all company employees, which are handled by the department database, are correctly classified as employees.
7. The class $\texttt{Position}_2$ is disjoint to all other classes.

These extensional relations are detected partly by inspecting the schema, and partly by analyzing current database instances and organizational laws of the company.

We will use this small example to discuss the problems arising with using binary extensional assertions only, which is proposed by some integration methods.

As a first observation, we realize that conditions 5 and 6 allow only for certain overlaps between the extensions of $\texttt{Employee}_1$, $\texttt{Employee}_2$ and $\texttt{Customer}_2$. These overlaps are graphically depicted in Figure 3.2 (using the well-known Euler/Venn diagram notation).



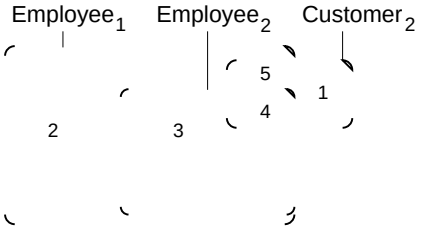
Local Class	1	2	3	4
$\texttt{Employee}_1$		√	√	√
$\texttt{Employee}_2$			√	√
$\texttt{Customer}_2$	√			√

Figure 3.2: Exact extensional relationships among $\texttt{Employee}_1$, $\texttt{Employee}_2$, and $\texttt{Customer}_2$

On the right side of Figure 3.2, we have shown an equivalent notation for extensional overlaps which does not show the intrinsic restrictions of a diagrammatic notation. Such

an *extension diagram* splits all extensions into *base extensions* (labeled with numbers 1 to 4 in this example) and specifies the relation between class extensions and base extensions as a Boolean matrix.

In the case of using only *binary* extensional assertions, we can only model the subset relationships between $\texttt{Employee}_2$ and $\texttt{Employee}_1$ and between $\texttt{Employee}_1$ and $\texttt{Customer}_2$, respectively. This situation is depicted in Figure 3.3. The resulting diagrams show an additional base extension (5), which is as a result of our extensional analysis always empty (condition 6 of our extensional condition list).



Local Class	1	2	3	4	5
$\texttt{Employee}_1$		$\surd$	$\surd$	$\surd$	$\surd$
$\texttt{Employee}_2$			$\surd$	$\surd$	
$\texttt{Customer}_2$	$\surd$			$\surd$	$\surd$

Figure 3.3: Extensional relationships among $\texttt{Employee}_1$, $\texttt{Employee}_2$, and $\texttt{Customer}_2$ due to binary assertions

As a consequence of this example, we state that using binary assertions only is not sufficient for an exact extensional analysis. Besides the GIM approach, only few other approaches [CL93, Dup94] allow for extensional assertions of higher order than binary:

- [CL93] allows for arbitrary assertions specified in description logic but transform them into a graph supporting binary assertions only.
- [Dup94] uses a fixed set of special assertions like $A \cup B = C$ which is too restricted in expressibility.

Chapter 4

The GIM Method

One main problem of database integration is the complexity of resolving schema heterogeneity. This complexity increases with the number of (non-orthogonal) modeling concepts offered by a database model. We propose therefore a minimal model having only few orthogonal modeling concepts to reduce this complexity.

4.1 Basic Ideas of the GIM Method

For the integration process we use the data model *GIM (Generic Integration Model)* as intermediate representation for the schemata S^C, S^H and S^I. GIM is a class-based data model but does only support disjoint extensions, i.e., GIM does not support subclass relationships as result of specializations. This restriction allows a resolution of most conflicts as part of the schema translation.

The construction of external views can be supported by design algorithms. We propose to adapt algorithms from the mathematical field of *formal concept analysis* for deriving external views. This idea is depicted in Figure 4.1. As result of using these al-

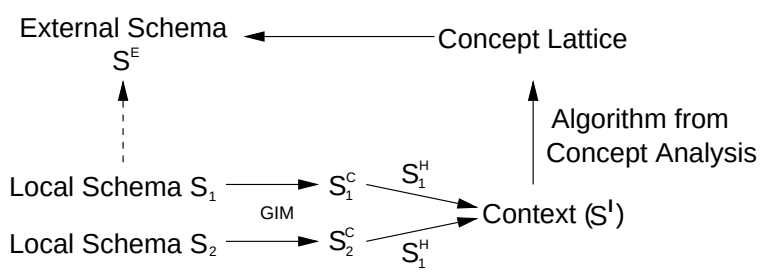


Figure 4.1: Steps of the GIM method

gorithms, we construct a correct (and minimal) external schema S^E in an object-oriented database model with a specialization hierarchy. Based on this first result, we may adapt the external schema to specific requirements (i.e., no multiple inheritance) or translate

it to other data models. The mapping information needed for query translation is a by-product of the generation.

Following [SCG91] GIM can be classified as a semantically poor integration model. However, the derived external schema is presented in a semantically rich database model. The intermediate schemata have not to be understandable by users and can be hidden as intermediate representations in a design tool. The tool **SIGMA**$_{Bench}$ based on these principles is discussed later in Section 9. As a result, GIM schemata have not to be understandable (and can become very large because of generating a high number of disjoint extensions). However, for presenting the integration steps we will use a graphical notation for GIM in the following sections.

The following subsection formalizes the GIM data model.

4.2 Definition of GIM

GIM is a class-based data model supporting classes with *disjoint* extensions. Attributes have simple atomic data types. Because of these simple structures, integrity constraints, for example uniqueness constraints and range restrictions, play an important role in modeling semantic information. Relationships are expressed by bi-directional reference attributes. Uni-directional references are not supported.

In the following we use some mathematical conventions. A function is considered as a set of ordered pairs of elements which assigns to each element of the domain (abbreviated by *dom*) exactly one element of the range. A special function is a set-valued function. Its range contains sets.

We assume that the set $\mathcal{D}at$ and the set function $\mathcal{D}omain$ are given. $\mathcal{D}at$ contains the names of elementary data types and reference datatypes for references to one or more classes. The set-valued function $\mathcal{D}omain$ is a function over $\mathcal{D}at$. $\mathcal{D}omain$ assigns a set of possible values to each data type name. A value of a reference data type is a set of object identifiers of objects from the referenced classes.

The constituents of GIM are now defined as follows:

- Class $C \stackrel{def}{=} (Cname, Att, IC)$.

  $Cname$ is the name of the class and Att is a partial function from its defined attribute names into $\mathcal{D}at$ defining the intension of a class. Each class has an attribute Id which represents the object identifiers. Att can also contain reference attributes. Additionally, IC contains integrity constraints like local uniqueness constraints, range restrictions and cardinality constraints.

- Schema $S \stackrel{def}{=} (Sname, \mathcal{C}, Unique)$.

  $Sname$ is the name of the schema and $\mathcal{C}$ is the set of classes $\{C_1, C_2, \ldots, C_n\}$. $Unique$ is the 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. The object identifiers have to be unique.

  We have the following restrictions:

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

  and

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

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

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

  With $Ext^t_{Id,C}$ we denote 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.[1] A state assigns to each class of the schema its extension:

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

  Additionally, we require:

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

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

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

- Additionally, all constraints have to be satisfied by a state. We use $Dom^S_C(A)$ for denoting the range of attribute A of class C in schema S under consideration of all constraints.

Graphical Presentation

For discussing 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 as $\bigcup_{C \in S.\mathcal{C}} C.Att$. The type of a reference to extension n is denoted as En.

The horizontal dimension is defined by the set of classes, which are called base extensions. These extensions are disjoint and used as placeholders for concrete objects.

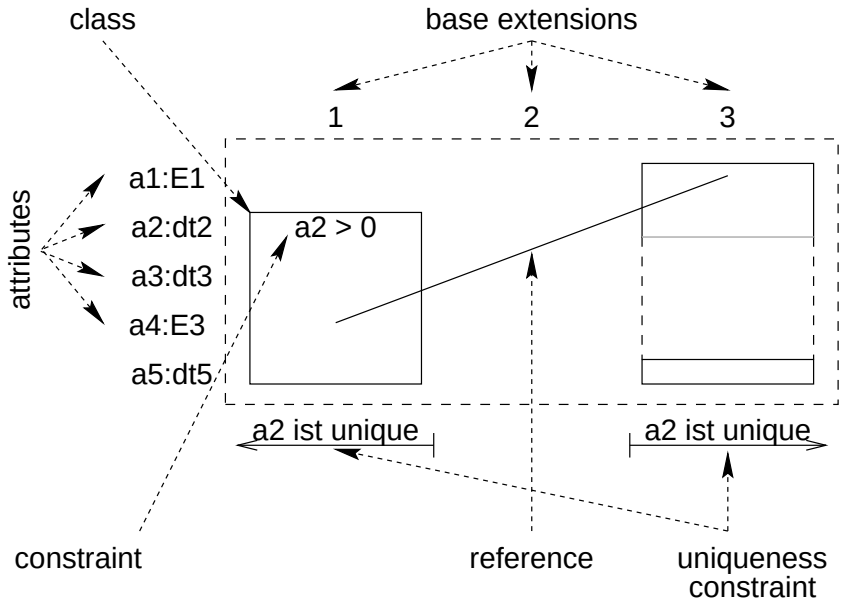


Figure 4.2: Graphical GIM Representation

For simplicity, base extensions are identified by numbers because there is no need to find intuitive names for them.

Figure 4.2 shows the two-dimensional diagram defining a GIM schema. References are denoted by a line between the two paired inverse reference attributes. Classes are represented as rectangles grouping attributes for an extension. The figure shows additionally the representation of local constraints and global uniqueness constraints.

Later on we will use a further simplified notation as tables, for example in Table 5.2. Such a table consists of attribute names, numbered base extensions, '$\surd$'-entries for the attribute-extension relation, and shadowed areas for constraints.

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

Chapter 5

Schema Translation

After defining the Generic Integration Model we will discuss the translation of local schemata into this data model. Since the translation is not the focus of our paper and since there are many translation operations dependent from the source data model we will sketch main operations only.

In contrast to many other database models GIM supports bi-directional reference attributes only and disjoint class extensions. Furthermore, except the reference data type, GIM requires atomic attribute types.

Bi-directional reference attributes do not prefer a certain direction and are therefore, in this sense, independent of a particular view. An inverse reference attribute can easily be created to a given uni-directional one. Such an inverse attribute is created for each class the given reference attribute points to. On data level, for each reference from an object to another object we have the inverse reference simultaneously. The consistency of this redundancy is guaranteed by the Generic Integration Model. Therefore, we have a bijective database mapping after creating inverse reference attributes for uni-directional attributes. In our example the attribute `job` of the class $\texttt{People}_2$ is a uni-directional reference attribute pointing to the class $\texttt{Position}_2$. The new inverse reference attribute is the attribute `people` of the class $\texttt{Position}_2$ pointing to class $\texttt{People}_2$.

GIM schemata are always in the 1NF. Furthermore, it does not support functional dependencies except uniqueness constraints which represent funcional dependencies from attribute combinations to sets of object identifiers. The relational normalization theory [Ull89] gives us a formal framework to transform schemata into 1NF schemata and uniqueness constraints. During the transformation a subset of attributes of a relation often becomes a new relation. Since the Generic Integration Model supports the idea of objects, new object identifiers must be assigned to the tuples (objects) of the new relations (classes). In order to guarantee a bijective database mapping, the mapping between new objects and assigned object identifiers must be stored by the FDBS. New relations (classes) are linked to the orignal ones by new bi-directional reference attributes using the new object identifiers. Our example schemata are in the 1NF and need not to be transformed.

Class extensions of a GIM schema are always mutually disjoint. This property is usually not fulfilled in other data models. Specialization in object-oriented database models, for example, means subset relations between class extensions. Specialization can also exist in relational schemata by using primary and foreign keys. A decomposition of overlapping class extensions provides disjoint extensions. For subset relationships in specialization hierarchies it is sufficient to compute shallow class extensions. A shallow extension of a class consists of instances which are not instances of its subclasses. For the bijective database mapping φ^C the FDBS must store the information about the decomposition.

In a schema a reference attribute can point to a class which have to be decomposed due to the requirement of disjoint class extensions. After the decomposition this attribute points to the new classes. This is the reason why GIM allows reference attributes pointing to more than one class. We will later show how class extensions are composed again.

Integrity constraints of a class to be decomposed can exist. Range restrictions are 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 Figure 5.1 and 5.2. These GIM schemata in the reduced form as tables are shown in Table 5.2. The mappings between the original classes and the GIM classes are presented in Table 5.1.

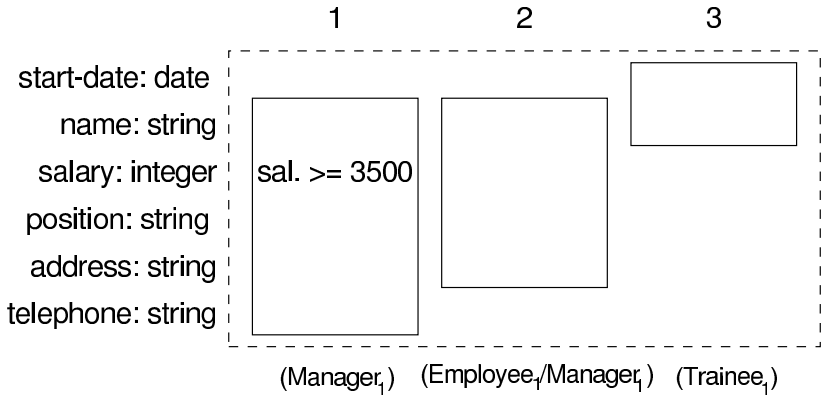


Figure 5.1: Graphical GIM-representation of schema 1

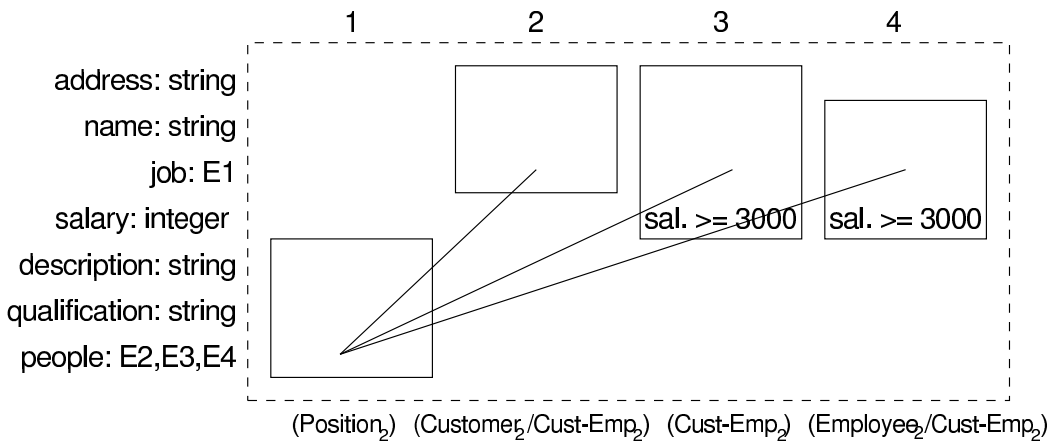


Figure 5.2: Graphical GIM-representation of schema 2

local class	base extension
Person_1	1,2,3
Employee_1	1,2
Trainee_1	3
Manager_1	1

local class	base extension
Position_2	1
People_2	2,3,4
Customer_2	2,3
Employee_2	3,4
Cust-Emp_2	3

Table 5.1: S1 and S2: Extensional mapping

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	√			

Table 5.2: S1 and S2 as GIM tables

Chapter 6

Schema Homogenization

This section describes how to homogenize the component schemata S_1^C and S_2^C. Homogenizing these schemata produces the GIM schemata S_1^H and S_2^H and their corresponding database mappings φ_1^H and φ_2^H. As mentioned in Chapter 2, 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 conflict classes in the following sequence:

1. *Structure conflicts:* correspondences between attributes and classes due to correspondences between attribute values and objects;

2. *Attribute conflicts:* correspondences between attributes due to correspondences between their attribute values;

3. *Extensional conflicts:* correspondences among classes due to potential redundancy among their extensions.

The list does not explicitly contain name conflicts between attributes and classes, respectively, e.g. synonyms and homonyms. Resolving attribute name conflicts is contained within resolving the attribute conflict and resolving class names conflicts is contained within the extensional conflict resolution.

The three steps of homogenization produce intermediate schemata and the database mappings $\varphi^{HS}, \varphi^{HA}$, and φ^{HE} where φ^H is built by $\varphi^{HE} \circ \varphi^{HA} \circ \varphi^{HS}$. Following each of the step means firstly detecting conflicts and then to resolve them.

6.1 Homogenization: Intensional Aspects

This subsection describes how to deal with the structure and the attribute conflicts.

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 class $\texttt{Manager}_1$ and the class $\texttt{Position}_2$ (cf. Figure 6.1).

To find all structure conflicts the designer has to compare each attribute with each class. 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 (cf. for example [BH91]).

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

Schema Transformation: Here we assume the attribute $\mathrm{A}_1 \ldots \mathrm{A}_n$ of the classes $\mathrm{C}_1 \ldots \mathrm{C}_n$ of the first schema with

$$Dom_{C_1}^{S_1^C}(\mathrm{A}_1) = ... = Dom_{C_n}^{S_1^C}(\mathrm{A}_n)$$

are in conflict with the class C of the second schema. For resolving the structure conflict a new class C_{new} for the first schema has to be created. The name of the new class is adopted from the class C. The class C_{new} has exactly one non-reference attribute A_{new} with the name of the corresponding attribute of the class C. The datatype is adopted from the attributes $\mathrm{A}_1 \ldots \mathrm{A}_n$. An additional attribute is a reference attribute A_R pointing to the classes $\mathrm{C}_1 \ldots \mathrm{C}_n$. The original attributes $\mathrm{A}_1 \ldots \mathrm{A}_n$ are now transformed to reference attributes inverse to A_R pointing to the class C_{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_{new}, corresponding integrity constraints are copied to the new class C_{new}. Additionally, a uniqueness constraint on attribute A_{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 $\mathrm{A}_1 \ldots \mathrm{A}_n$ is replaced by a null reference. Therefore, the attribute A_{new} must be a not null attribute. Furthermore, the class C_{new} can contain only such objects for which corresponding values of the original attributes 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 pictured in Figure 6.2.

Mapping Database States: The function φ^{HS} maps the database states according to the schema transformation. As stated in Section 2 the function φ^{HS} must be a bijective function. For each occurring value of an attribute being involved in a structure conflict a new objects must be created and inserted into the corresponding new class. 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 and guarantees bijectivity and a stable mapping. Furthermore, the reference attributes must be set to the correct object identifier.

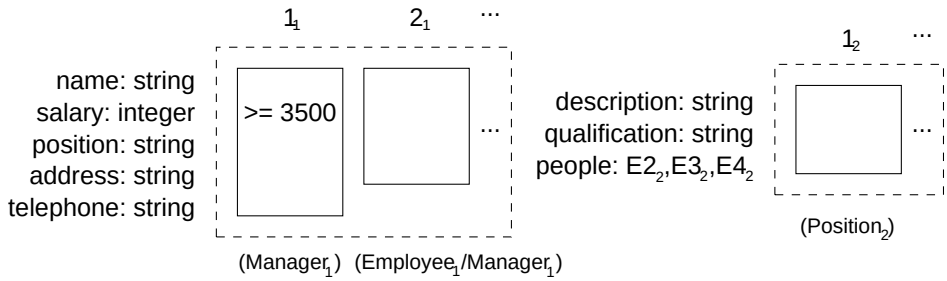


Figure 6.1: Structure conflict between the example schemata

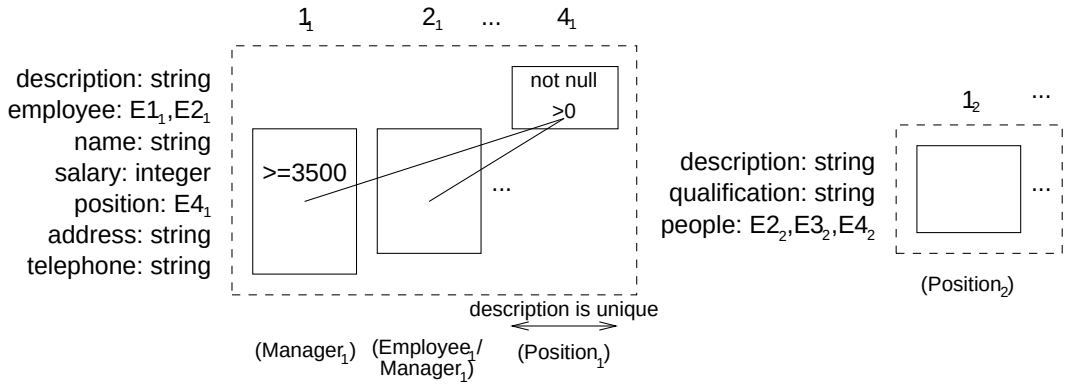


Figure 6.2: Resolution of a structure conflict

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 due to their related values. We consider here conflicts between non-reference attributes only. Resolving reference attribute conflicts would require a specialization on references which is supported by very few database models.

Our 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 help 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. [BH91]).

Due to the demand for minimality and understandability the problem of different representations of one property must be resolved. Here we assume the attribute A_1 of the class C_1 is in conflict with the attribute A_2 of the class C_2 and

$$\alpha \subseteq Dom_{C_1}^{S_1^C}(A_1) \times Dom_{C_2}^{S_2^C}(A_2)$$

relates their attribute values to each other. The relation α can be specified as an explicit table or as a piece of code. In our example we can easily specify α by an algorithm which divides a value of $\texttt{name}_2$ into the first and the last name and then compares the last name with a value of $\texttt{name}_1$.

A conflict exists if α is not the identity function. We distinguish three types of attribute conflicts. Of course, they can occur in combination:

- *Missing injectivity in one direction:*

$$\exists (a_1, b_1), (a_2, b_2) \in \alpha : (a_1 \neq a_2 \wedge b_1 = b_2) \vee (a_1 = a_2 \wedge b_1 \neq b_2)$$

- *Different attribute domains:* $Dom_{C_1}^{S_1^C}(A_1) \neq Dom_{C_2}^{S_2^C}(A_2)$
- *Different values:* $\exists (a, b) \in \alpha : a \neq b$

Schema Transformation due to Missing Injectivity: This conflict produces a loss of information if due to α attribute values are mapped into another representation and then re-mapped into the original representation. Such a cyclic mapping is necessary if you want to move a new value from the integrated schema to a local database and then to read this value again.

Here we assume to have missing injectivity between attribute A_1 and A_2. If we have missing injectivity in both directions, then the conflict must be resolved twice. The main idea to resolve the conflict is to split attribute A_1 into two new attributes A_1' and A_2'. The name of A_1' is adopted from attribute A_2. The designer has to specify the name of A_2'. Furthermore, the datatypes and integrity constraints for A_1' and A_2' must be specified by the designer, too.

In our example, the mapping between $A_1 = \texttt{name}_2$ to $A_2 = \texttt{name}_1$ is not injective. We split the attribute $A_1 = \texttt{name}_2$ into an attribute $A_1' = \texttt{name}_2$ and $A_2' = \texttt{first-name}_2$ which contain last and first names, respectively. Now, there is an injective mapping from attribute $A_1' = \texttt{name}_2$ to $A_2 = \texttt{name}_1$ and no correspondence between $A_2' = \texttt{first-name}_2$ and $A_2 = \texttt{name}_1$. Table 6.1 shows the splitting of some example values whereas Figure 6.3 depicts the modified schema.

$\mathbf{A}_1$= name$_2$	$\mathbf{A'}_1$= name$_2$	$\mathbf{A'}_2$= first-name$_2$	$\mathbf{A}_2$= name$_1$
Max Hotzenplotz	Hotzenplotz	Max	Hotzenplotz
Emil Hotzenplotz	Hotzenplotz	Emil	Hotzenplotz
Franz Klammer	Klammer	Franz	Klammer

Table 6.1: Splitting of Attribute name$_2$ and related Attribute name$_1$

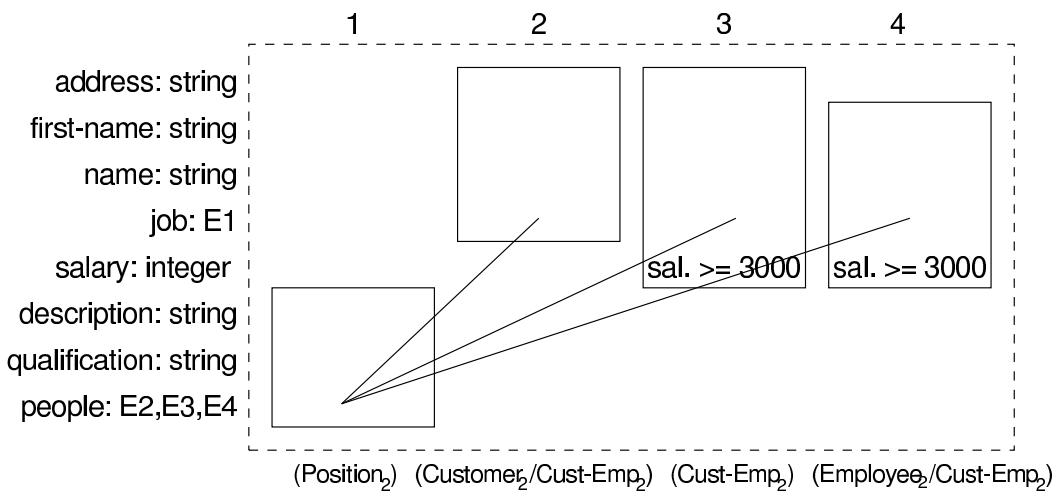


Figure 6.3: Resolution of an attribute conflict in Schema 2

Mapping Database States due to Missing Injectivity: For a correct splitting of the attribute A_1 there must exist a *bijective* function

$$\varsigma \subseteq Dom_{C_1}^{S_1^C}(\mathrm{A}_1) \times (\mathrm{X}_1 \times \mathrm{X}_2),$$

where X_1, X_2 are the domains of the new attributes ($Dom_{C_1}^{S_1^C}(\mathrm{A}'_1)$ =X_1 and $Dom_{C_1}^{S_1^C}(\mathrm{A}'_2)$ =X_2). In the example an algorithm splits a full name into first and last name.

From the function ς we derive the functions ς_1 and ς_2:

$$\begin{aligned} \varsigma_1 &= \{(x, y_1) | (x, (y_1, y_2)) \in \varsigma\} \\ \varsigma_2 &= \{(x, y_2) | (x, (y_1, y_2)) \in \varsigma\} \end{aligned}$$

To guarantee a correct value mapping the functions ς_1, and ς_2 and with that also ς must fulfill several requirements:

- ς_1 must cause an injective mapping from A'_1 to A_2:

$$\forall (x_1, y_1), (x_2, y_2) \in \alpha : y_1 = y_2 \Rightarrow \varsigma_1(x_1) = \varsigma_1(x_2)$$

- ς_1 must reserve distinguishable values:

$$\forall a, b, \in dom(\varsigma) : \forall (a, y_1), (b, y_2) \in \alpha : y_1 \neq y_2 \Rightarrow \varsigma_1(a) \neq \varsigma_1(b)$$

- There must not exist a semantic relation between A_1', A_2', and A_2', A_2, respectively.

For the correct value mapping the federated database system has to manage the functions $\varsigma^{-1}, \varsigma_1$, and ς_2. They can be stored as explicit tables or as pieces of code.

The problem is now to find the function ς fulfilling the requirements above. The following discussion is intended to help the designer finding ς:

The relation α induces an equivalence relation F on attribute A_1:

$$xFy \Leftrightarrow \exists z : (x, z), (y, z) \in \alpha$$

The function ς_1 is derived from F. It assigns a value representing an equivalence class to a value of A_1:

$$\varsigma_1(x) = [x]_F$$

It can be proven easily that ς_1 produces injectivity between A_1' and A_2. Applying this approach to our example (cf. Table 6.1) produces two equivalence classes:

$$\begin{aligned} [1] = &\ \{\texttt{Max Hotzenplotz,Emil Hotzenplotz}\} \\ [2] = &\ \{\texttt{Franz Klammer}\} \end{aligned}$$

The equivalence classes are represented by the common element which is in that case the last name.

The next problem is to find ς_2. The goal is to find a feature which makes the elements within each equivalence class distinguishable and independent from the values of ς_1. If there is an order on $Dom_{C_1}^{S_1^C}(A_1)$ then the elements within each equivalence class can be numbered. The function ς_2 assigns then numbers to values of A_1. In our example, the elements within an equivalence class can be distinguished by their first names. Therefore, ς divides a full name into its first name and last name.

Schema Transformations due to Different Attribute Domains and Values: After resolving missing injectivity between related attributes different domains and values can occur. The basis to resolve these conflicts is again the relation α. The relation α between attributes involved in a conflict of missing injectivity can be easily reconstructed using ς and the original relation α.

The demand for minimality and understandability requires the resolution of conflicting domains of semantically related attributes. The main idea is to map semantically related attribute values to same values of a new datatype. The new datatype itself must be the same in both schemata. In order to guarantee surjective mappings different sets of integrity constraints can be specified on the new datatype. Therefore, the designer has to specify a new datatype and integrity constraints. On schema level the definition of the related attributes must now be replaced by the new datatype and integrity constraints. The names of the changed attributes must be equal. Same names for attributes being not semantically related must be changed in that way that same names express only semantically related attributes.

Mapping Database States due to Different Attribute Domains and Values: Assume the different domains of the new datatype restricted by different integrity constraint sets are X_1 and X_2. The designer has now to specify two *bijective* mappings[1]:

$$\eta_1 \subseteq Dom_{C_1}^{S_1^C}(A_1) \times X_1 \text{ and } \eta_2 \subseteq Dom_{C_2}^{S_2^C}(A_2) \times X_2$$

The mappings must reserve the relations between the original attribute values:

$$\forall x \in dom(\eta_1) : \forall y \in dom(\eta_2) : (x, y) \in \alpha \Leftrightarrow \eta_1(x) = \eta_2(y)$$

A conflict of different attribute values occurs for example if in one database the salary is stored in Dollars whereas the other database uses Deutsche Mark. The designer can now decide to map the different currencies into Euro.

A further example of different attribute values can appear in our introduced example. Job descriptions can be different in the given schemata. In that case, the mappings η_1 and η_2 must be specified by an explicit table.

Due to the bijective mappings of ς, η_1, and η_2 the mapping φ^{HA} is a bijective mapping of database states, too.

[1] One of the mappings is frequently the identity function.

6.2 Homogenization: Extensional Aspects

The Generic Integration Model requires disjoint class extensions. Whereas class extensions within one component database already meet this requirement[2] this must not be the case if we compare class extensions from different databases. This aspect is the topic of this subsection.

Before we can merge the schemata into one GIM schema the problem of extensional overlaps expressing redundancy must be resolved. The goal is to have identical or disjoint class extensions only. As a prerequisite we need correct information about extensional relations among class extensions. Extensional relations can be regarded as global integrity constraints hidden in application semantics. They can be expressed by an Euler/Venn-diagram or by an extension diagram (cf. Section 3). Different sources for extensional relations can be exploited:

1. *Information from the given schema and data model:* From 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, additionally constraints can state exclusive or total specializations. Many data models require disjoint extensions for all classes which are not in a specialization relation. The schema and data model sources, however, have a limitation. They express information about extensional relations only within a single database.

2. *Database states:* The designer has often access to states of the databases to be integrated. Comparing available class extensions between databases can help to detect redundancy. Such a redundancy excludes a disjoint relation between the corresponding classes. The database state source has a limitation, too. From finding disjoint class extensions in a given database state the designer cannot conclude disjoint class extensions valid at any instant of time.

3. *Integrity constraints:* Comparing integrity constraints can help to detect extensional relations among classes from different schemata. If, for example, no database state can simultaneously fulfill the combined integrity constraints of different classes, then a disjoint extensional relation can be concluded. For more details see [RR97, TS98, TS99].

4. *Background knowledge of the designer:* In practical scenarios the information coming from the previous three sources is often not sufficient. The designer has to use his background knowledge in order to specify complete information about extensional relations. Therefore, the complete specification of extensional relations usually requires human interactions.

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

[2]Disjoint class extensions result from the translation step described in Section 5.

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. [SPD92], 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. Of course, only database states at same instants of time are considered. Very important for the schema integration is information about the extensional development of databases in the future. One extensional assertion, for example, can state, that the extension of the class $\texttt{Employee}_2$ is *always* a subset of the extension of class $\texttt{Employee}_1$.

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 Section 1. Between two classes C_1 and C_2 the following extensional assertions can be defined:

$$\begin{aligned}
\textit{Disjointness}&: C_1 \varnothing C_2 :\Leftrightarrow \forall t : \forall id_1 \in Ext^t_{Id,C_1} : \forall id_2 \in Ext^t_{Id,C_2} : \neg\text{same}(id_1, id_2)\\
\textit{Containment}&: C_1 \subseteq C_2 :\Leftrightarrow \forall t : \forall id_1 \in Ext^t_{Id,C_1} : \exists id_2 \in Ext^t_{Id,C_2} : \text{same}(id_1, id_2)^3\\
\textit{Equivalence}&: C_1 \equiv C_2 :\Leftrightarrow (C_1 \subseteq C_2) \wedge (C_2 \subseteq C_1)\\
\textit{Overlap}&: C_1 \Cap C_2 :\Leftrightarrow \text{otherwise}
\end{aligned}$$

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 stated if no other assertion can be stated.

In Section 3 we motivated the need for extensional assertions between more than two classes. Therefore, we enhance extensional assertions to allow comparisons between *set expressions* on class extensions using the operations union, difference and intersection (cf. [ST98]).

The extensional relations for our example are informally given in Section 3. We implicitly assume overlapping class extensions among classes from different databases if no other extensional assertion was stated. The following list contains the explicit extensional assertions:

$\texttt{Employee}_1 \subseteq \texttt{Person}_1$, $\texttt{Manager}_1 \subseteq \texttt{Employee}_1$, $\texttt{Trainee}_1 \subseteq \texttt{Person}_1$,

$\texttt{Customer}_2 \subseteq \texttt{People}_2$, $\texttt{Employee}_2 \subseteq \texttt{People}_2$, $\texttt{Cust-Emp}_2 \subseteq \texttt{Customer}_2$,

$\texttt{Cust-Emp}_2 \subseteq \texttt{Employee}_2$,

$\texttt{Person}_1 \equiv (\texttt{Employee}_1 \cup \texttt{Trainee}_1)$, $\texttt{Employee}_1 \varnothing \texttt{Trainee}_1$,

$\texttt{People}_2 \equiv (\texttt{Customer}_2 \cup \texttt{Employee}_2)$, $\texttt{Cust-Emp}_2 \equiv (\texttt{Customer}_2 \cap \texttt{Employee}_2)$,

$\texttt{Employee}_2 \subseteq \texttt{Employee}_1$,

$(\texttt{Employee}_1 \cap \texttt{Customer}_2) \subseteq \texttt{Employee}_2$,

$\texttt{Position}_2 \varnothing \texttt{Person}_1$, $\texttt{Position}_2 \varnothing \texttt{People}_2$,

[3] Please note the injectivity of the relation 'same' which follows from transitivity and unique object identifiers.

The assertions above are defined on local classes which were translated to GIM classes. Table 5.1 contains information how the local classes are mapped to GIM classes. Considering these mappings we obtain the following extensional assertions:

$$1_1 \cup 2_1 \subseteq 1_1 \cup 2_1 \cup 3_1 \quad (6.1)$$
$$1_1 \subseteq 1_1 \cup 2_1 \quad (6.2)$$
$$3_1 \subseteq 1_1 \cup 2_1 \cup 3_1 \quad (6.3)$$

$$2_2 \cup 3_2 \subseteq 2_2 \cup 3_2 \cup 4_2 \quad (6.4)$$
$$3_2 \cup 4_2 \subseteq 2_2 \cup 3_2 \cup 4_2 \quad (6.5)$$
$$3_2 \subseteq 2_2 \cup 3_2 \quad (6.6)$$
$$3_2 \subseteq 3_2 \cup 4_2 \quad (6.7)$$

$$1_1 \cup 2_1 \cup 3_1 \equiv 1_1 \cup 2_1 \cup 3_1 \quad (6.8)$$
$$1_1 \cup 2_1 \varnothing 3_1 \quad (6.9)$$

$$2_2 \cup 3_2 \cup 4_2 \equiv 2_2 \cup 3_2 \cup 3_2 \cup 4_2 \quad (6.10)$$
$$3_2 \equiv (2_2 \cup 3_2) \cap (3_2 \cup 4_2) \quad (6.11)$$

$$3_2 \cup 4_2 \subseteq 1_1 \cup 2_1 \quad (6.12)$$
$$(1_1 \cup 2_1) \cap (2_2 \cup 3_2) \subseteq 3_2 \cup 4_2 \quad (6.13)$$
$$1_2 \varnothing 1_1 \cup 2_1 \cup 3_1 \quad (6.14)$$
$$1_2 \varnothing 2_2 \cup 3_2 \cup 4_2 \quad (6.15)$$

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

$$\texttt{Position}_1 \varnothing \texttt{Person}_1, \texttt{Position}_1 \varnothing \texttt{People}_2,$$
$$\texttt{Position}_1 \subseteq \text{Position}_2$$

With respect to the translation to GIM classes we obtain:

$$4_1 \varnothing 1_1 \cup 2_1 \cup 3_1 \quad (6.16)$$
$$4_1 \varnothing 2_2 \cup 3_2 \cup 4_2 \quad (6.17)$$
$$4_2 \subseteq 4_1 \quad (6.18)$$

All classes within a GIM schema are extensionally mutually disjoint. This implicit extensional assertion makes the assertions (1) to (11), (15), and (16) obsolete. Please note, if there is no explicit assertion between two class extensions from different databases then they overlap.

The next step is the derivation of an extension diagram from a set of extensional assertions. [ST98] describes an suitable algorithm for that problem. The algorithm computes an extension diagram with a minimal number of base extensions by exploiting simplification rules of propositional logic. Table 6.2 shows the computed extension diagram for our example. By using Tables 6.2 and 5.1 an extension diagram for the original classes can be derived (see Table 6.3).

Schema Transformation: An extension diagram defines a set of base extensions. Each component class extension is fixed by the union of corresponding base extensions.

S^C	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
$\mathbf{1}_1$	√	√	√								
$\mathbf{2}_1$				√	√	√					
$\mathbf{3}_1$							√	√			
$\mathbf{4}_1$										√	√
$\mathbf{1}_2$											√
$\mathbf{2}_2$							√		√		
$\mathbf{3}_2$			√	√							
$\mathbf{4}_2$		√			√						

Table 6.2: Extension Diagram for S_1^C and S_2^C

S	S^C	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
`Person`$_1$	$\mathbf{1}_1 \cup \mathbf{2}_1 \cup \mathbf{3}_1$	√	√	√	√	√	√	√	√			
`Employee`$_1$	$\mathbf{1}_1 \cup \mathbf{2}_1$	√	√	√	√	√	√					
`Manager`$_1$	$\mathbf{1}_1$	√	√	√								
`Trainee`$_1$	$\mathbf{3}_1$							√	√			
`Position`$_1$	$\mathbf{4}_1$										√	√
`People`$_2$	$\mathbf{2}_2 \cup \mathbf{3}_2 \cup \mathbf{4}_2$		√	√	√	√		√		√		
`Customer`$_2$	$\mathbf{2}_2 \cup \mathbf{3}_2$			√	√			√		√		
`Employee`$_2$	$\mathbf{3}_2 \cup \mathbf{4}_2$		√	√	√	√						
`Cust-Emp`$_2$	$\mathbf{3}_2$			√	√							
`Position`$_2$	$\mathbf{1}_2$											√

Table 6.3: Extension Diagram for S_1 and S_2

Local Class	1	2	3	4	5	6	7	8	10	11
description									√	√
employee									√	√
start-date							√	√		
name	√	√	√	√	√	√	√	√		
salary	√	√	√	√	√	√				
position	√	√	√	√	√	√				
address	√	√	√	√	√	√				
telephone	√	√	√							

Table 6.4: Example: S_1^H

Local Class	2	3	4	5	7	9	11
address		√	√		√	√	
first-name	√	√	√	√	√	√	
name	√	√	√	√	√	√	
job	√	√	√	√	√	√	
salary	√	√	√	√			
description							√
qualification							√
people							√

Table 6.5: Example: S_2^H

Resolving extensional conflicts means extensionally decomposing classes into smaller classes. Each new class represents exactly one base extension. 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.

The decomposition of classes follows the same procedure as described for schema translation step (cf. Section 5). As result we obtain the homogenized schemata S_1^H and S_2^H.

The homogenized schemata of our example are presented in Table 6.4 and Table 6.5. The shaded areas indicate the existence of integrity constraints (cf. Table 6.6).

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 extensions from different databases. For example, base extension 2 contains all manager objects of the

GIM Class	Integrity Constraint
1,2,3	`salary` ≥ 3500
2,3,4,5	`salary` ≥ 3000
10,11	`description` is not null
10,11	card(`employee`) > 0
10∪11	unique(`description`)

Table 6.6: Integrity Constraints

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. We assume here the existence of such a relation. Producing and managing such a relation is a very hard job but not focus of this work. Instead, for that problem we refer to [Pu91, LSPR93, SS95, ZHKF95, CTK96].

By use of an extension diagram for each base extension a producing set expression can easily be found. Therefore, an extension diagram exactly defines the mapping of component classes into smaller ones and vice versa. Furthermore, the relation `same` is a one-to-one-relation. For these reasons, the database mapping function φ^{HE} is bijective. Since the mappings φ^{HE}, φ^{HA}, and φ^{HS} are bijective mappings the mapping φ^{H} is bijective, too.

Chapter 7

Schema Merging

This design step merges the homogenized schemata S_1^H and S_2^H into the schema S^I. Between the homogenized schemata same class names indicate same extensions. Classes without an equivalent[1] class in the other schemata are copied into the integrated schema. Pairs of equivalent classes, however, are merged into one class, respectively. A class $C_1 \in S_1^H$ and a class $C_2 \in S_2^H$ with $C_1.Cname = C_2.Cname$ are merged into $C \in S^I$ with

$$\begin{aligned} C.Cname &= C_1.Cname \\ C.Att &= C_1.Att \cup C_2.Att. \end{aligned}$$

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:

- *Incomplete design:* The specified constraints are too weak in one schema or some constraints are simply forgotten;
- *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 [BC86, Con86, EJ95, EJ96], the detection of conflicting integrity constraints is in general an undecidable problem. Restricting the power of integrity constraints, however, makes the problem decidable (cf. [TS99]). In the GIM method, only very simple integrity constraints can be defined. We assume the designer is able to detect conflicting integrity constraints and to find conflict reasons. A wrong homogenization forces a redo of the homogenization. If, however, conflicts are caused by incomplete designs then different sets of integrity constraints are conjunctively combined for the integrated class. The combination is a conjunction since each instance of one class has

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

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:

$$\begin{aligned} C.IC &= C_1.IC \cup C_2.IC \\ S.Unique &= S_1^H.Unique \cup S_2^H.Unique \end{aligned}$$

Merging the homogenized schemata of our example produces an integrated schemata presented in Table 7.1. Table 6.6 shows the corresponding integrity constraints.

Mapping Database States: A mapping problem occurs due to semantic redundancy: The values of same attributes of classes with same names can differ. Various reasons can cause different values:

- *Timeliness:* Applications update values with different delays after a real-world object has been changed. In other words, different values denote real-world values at different instants of time.
- *Wrong data:* Often, a value does not correctly denote an attribute value of an real-world object. Reasons are, for example, misspellings, imprecise measurements, and bad chosen attribute domains.
- *Wrong homogenization:* Similar to conflicting integrity constraints, different values can be caused by wrong resolutions to attribute or extensional conflicts.

The reasons for differing values often occur in a combined fashion. Sometimes, combined integrity constraints help to detect wrong values. In general the designer cannot be totally sure that at least one value is correct.

Mapping database states means mapping a pair of attribute values to exactly one value for the integrated schema, respectively. Such a mapping is in general heavily dependent on the semantics of the attribute and the applications. Therefore, there is no standard solution to this problem. For each attribute A occurring in equally named classes the designer has to specify a function γ_A:

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

Depending on the quality of data, cf. [GS98], the function has to derive a value from a pair of given values. The following list shows some possible ways of computation:

- *First (second) value:* The function takes always the first (second) value since the designer knows that the first (second) database contains more correct values.
- *Aggregated value:* For some reasons, an aggregated value should be presented in the integrated schema. A typical aggregation is the average of both values.
- *Maximum (minimal) value:* Sometimes the designer knows, that the maximum (minimum) of both values should be chosen.

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											√

Table 7.1: Example – Integrated Schema

- *Dependency:* The decision on the right value can be dependent 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 current value.

If the input values of γ are equal, then the output value must be the same, too.

In general, the function γ is not injective and causes therefore a loss of information. If the designer does not want a loss of information then he must not treat those attributes as equal during the attribute conflict step.

For a non-injective function no inverse function can exist. In order to map a global value to the underlying databases a back function is required. The common back function is the identity function. More precisely, a value 'x' is mapped to the pair '(x,x)'.

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 [SS95] a bijective mapping between local and global mapping is needed.

The mapping function φ^I is a total and surjective function. Due to the missing injectivity of the functions γ the mapping φ^I is not injective, too.

Chapter 8

Deriving External Schemata

An integrated GIM schema S^I is not an appropriate schema for applications. Instead of being a schema for applications it allows a very elegant derivation of external schemata. If we look at an integrated schema, e.g. Table 7.1, we can make some interesting observations on which the derivation of external schemata bases:

1. *Rectangles represent classes:* In the schema you can find rectangles. A rectangle means here that for a certain set of base extensions 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 local classes as rectangles. For instance the local class $\texttt{People}_2$ is represented by the rectangle with the base extensions 2-5, 7, and 9 and the attributes `first-name`, `name`, and `job`.

2. *There are maximal rectangles:* Some rectangles can be augmented by base extensions or attributes. Rectangles which cannot be augmented are called *maximal* rectangles. The local class $\texttt{People}_2$, for example, can be augmented by the attribute `address` and is therefore not a maximal rectangle.

   Deriving external schemata means finding all rectangles. Due to the demand for completeness each tick in the table must be contained in at least one rectangle. Finding rectangles which are not maximal means finding many rectangles. An extreme situation occurs if we consider each tick as a rectangle. Since we are interested to find an external schema with a minimal number of classes we are interested in all maximal rectangles of an integrated schema.

3. *Maximal rectangles can overlap:* Figure 8.1 shows an overlap of two rectangles. The rectangles are maximal rectangles. Thereby, if the extension of a class (rectangle) is a subset of the extension of an overlapping class then the attribute sets stand always in a superset relationship. Due to this observation, we can consider an overlap as a specialization between classes. The class with less extension is the subclass.

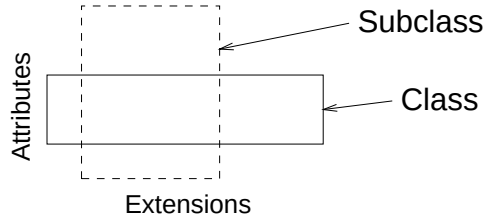


Figure 8.1: Overlapping maximal rectangles

This section contains two subsections. The first subsection shows how to derive a normalized external schema from an integrated schema. The second subsection explains the adaption of an external schema to a certain application view.

8.1 Automatic Derivation of a Normalized External Schema

Finding all maximal rectangles is very hard for a designer. For automation we propose to apply mechanisms of 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 [GW98] for more information about this theory.

The theory of concept analysis is based on the following formalization [Duq87]: A *context* (G, M, I) is given where G is a set of objects, M is a set of attributes (intension), and $I \subseteq G \times M$ is a binary relation between these (finite) sets. 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 $(A, B) \in \mathcal{P}(G) \times \mathcal{P}(M)$ 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) in (G, M, I), and let $\leq$ be the order relation on L defined by:

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

As result, we obtain a lattice denoted by:

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

The lattice operations are given by 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 the problem of deriving external schemata. A GIM-schema, e.g. expressed by Table 7.1, can be interpreted as a binary relation I, i.e. it represents a context.

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

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

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

A concept lattice derived from such a GIM-schema can then be considered as an external schema as follows:

- a concept $(A, B) \in L$ is a class with extension A and B defining its attributes;
- $\leq$ is the specialization relationship between two classes;
- $\wedge$ is the specialization operation (intersection of extensions) of two classes;
- $\vee$ is the generalization operation (intersection of attribute sets) of two classes;
- $(extent(M), M)$ is the bottommost class of the hierarchy (attribute set contains all attributes; extension may be empty);
- $(G, intent(G))$ 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 than 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.

Computationally Complexity: Unfortunately, this approach has two disadvantages. A context with $m := |G|$ and $n := |M|$ contains at most $2^{min(m,n)}$ concepts. An example of such a context is an '$n \times n$'-matrix in which each matrix element has a tick except the matrix elements on the main diagonal. The lattice from this context contains 2^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 disadvantage. It produces some concepts representing unnecessary classes. Consider the context depicted in Table 8.1.

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

Table 8.1: Context producing unnecessary Classes

The Hasse-diagrams of the resulting concept lattice are depicted in Fig. 8.2. The

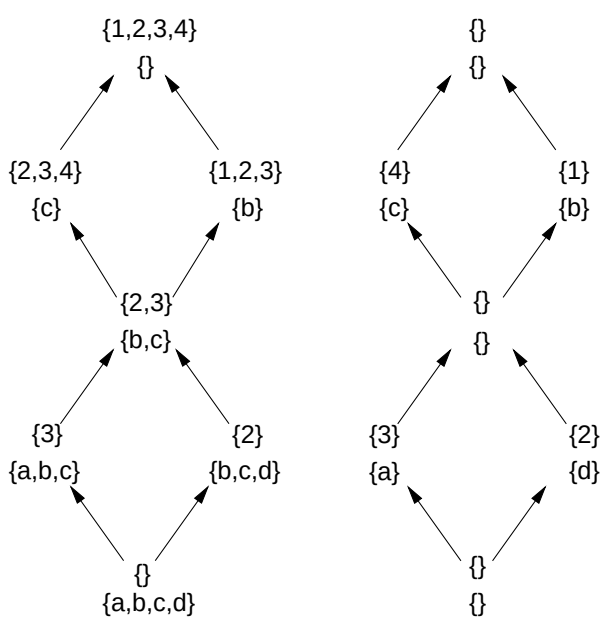


Figure 8.2: Resulting and reduced concept lattice

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 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 and can be omitted. Omitting such classes, however, destroys the lattice property. Therefore, we use the term 'specialization hierarchy' instead of lattice.

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

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

8.1.3 A Better Algorithm

In this subsection we present an algorithm which generates only valid classes. The algorithm computes a concept hierarchy instead of a concept lattice. The algorithm has the complexity $O(n^3)$. For each step of the following algorithm the complexity will be given. The input value for the complexity measurement is $n = max(|G|, |M|)$. G is the set of base extensions and M is the set of attributes. The complexity of a single step can be directly derived using the matrix representation as introduced in Table 7.1.

The input for the algorithm is a context (G, M, I). We use the functions *intent* and *extent* introduced in the previous section to define the sets Int and Ext. The sets Int and Ext contain the intents of each single base extension ($g \in G$) and the extents of each 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:

$$Con := Con_I \cup Con_E$$

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 M representing the irreflexive binary relation '<' defined on the generated classes of Con.

$$C_1 = (A_1, B_1) : C_2 = (A_2, B_2) :$$
$$C_1 < C_2 \Longleftrightarrow A_1 \subset A_2$$

A value '1' in M on row i and column j of M means that class C_i is a subclass of class C_j ($C_i < C_j$). 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 build this matrix is again $O(n^3)$. Computing the matrix needs comparisons between at most $2 * n$ classes with at most $2 * n$ classes. For each comparison at most n base extensions have to be examined.

The computation $M_N = M - (M \times M)$ 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 M_N defines the external classes with specialization relations.

For example, the new algorithm generates a concept hierarchy (cf. Figure 8.3) from the context depicted in Figure 8.1.

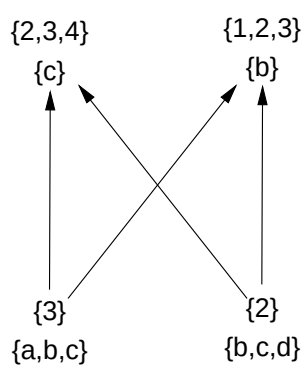


Figure 8.3: Resulting concept hierarchy

Theorem 8.1.1 *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 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) $\forall m_i^c \in [m_1^c, \ldots, m_b^c] : \exists g \in G :$
   $g \notin [g_1^c, \ldots, g_a^c] \wedge (g, m_i^c) \in I$
   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. □

Example: The integrated schema of our example presented in Figure 7.1 is regarded as a context. The improved algorithm computes the class set $Con = \{\texttt{C1}, \ldots, \texttt{C11}\}$. The extensions and attributes of these classes are presented in Table 8.2. Comparing the class extensions provides the matrix M in Table 8.3.

The matrix M_N without transitive subset relations is presented in Table 8.4.

8.1.4 Schema Derivation

From the class set Con and matrix M_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

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 8.2: Example – Global Classes

<	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
C1	0	0	0	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0	0	0
C3	1	0	0	0	0	0	0	0	0	0	0
C4	1	0	1	0	0	0	0	0	0	0	0
C5	1	0	1	0	0	0	0	0	0	0	0
C6	1	1	1	1	0	0	0	0	0	0	0
C7	1	0	1	1	1	0	0	0	0	0	0
C8	1	0	1	0	1	0	0	0	0	0	0
C9	1	0	1	1	1	0	1	1	0	0	0
C10	0	0	0	0	0	0	0	0	0	0	0
C11	0	0	0	0	0	0	0	0	0	1	0

Table 8.3: Matrix M

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 its name can be used.

The improved algorithm helps to compute an external schema as an object-oriented one. Of course, in some circumstances the designer wants to derive an external schema in another data model. In such cases, the set Con and the table M_N are very helpful, too. In the following we will concentrate only on object-oriented, external schemata.

Sometimes, the improved algorithm produces a loss of information. Therefore, we introduce the idea of a discriminant attribute, which is described in the next paragraph.

Up to now we have not considered integrity constraints yet. The paragraph after the next will explain how to deal with them.

<	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
C1	0	0	0	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0	0	0
C3	1	0	0	0	0	0	0	0	0	0	0
C4	0	0	1	0	0	0	0	0	0	0	0
C5	0	0	1	0	0	0	0	0	0	0	0
C6	-2	1	0	1	0	0	0	0	0	0	0
C7	-2	0	-1	1	1	0	0	0	0	0	0
C8	-1	0	0	0	1	0	0	0	0	0	0
C9	-4	0	-3	0	-1	0	1	1	0	0	0
C10	0	0	0	0	0	0	0	0	0	0	0
C11	0	0	0	0	0	0	0	0	0	1	0

Table 8.4: Matrix M_N

Discriminant: For deriving an external schema there is a requirement for avoiding unwanted loss of information. Sometimes, the proposed algorithm joins together base extensions to external classes thus from an external object we cannot conclude 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 and disjoint extensions 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 information loss we demand that for each object on the external level we are able to conclude its base extension from its state and class membership. This property is violated if the algorithm does not distinguish between base extensions, i.e. if the integrated schema contains base extensions with same attributes. Therefore, we introduce an artificial attribute called the *discriminant* for all base extensions $g \in G$ for which another 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 [GS91]). We have adopted the 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. If the discriminant does not carry semantics then its value is not available for an inserted object. In that case a designer specified default value for the discriminant determines the base extension and thereby the corresponding local classes.

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											√
discriminant		√	√	√	√						

Table 8.5: Example – integrated schema with discriminant

If you check the integrated schema in Table 7.1 you find the base extensions 2 and 3, and the base extensions 4 and 5 having same attributes, respectively. Therefore we introduce a discriminant for these base extensions. The discriminant determines whether a person is a customer or not. Table 8.5 presents the augmented integrated schema. Applying our improved algorithm again expands the classes C7 and C9 by the discriminant. The matrix M remains unchanged. Figure 8.4 depicts the corresponding external schema without class names but information about the class extensions. The extensional reduction produces the integrated schema depicted in Figure 8.5.

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 have no chance to understand when a global insertion fails due to constraint violations on local level. Please remember, a federated database system has to behave as a common database system for global applications.

The publications [CHS+97, CST98] 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 range restrictions and uniqueness constraints.

- *Range restrictions:* If all the base extensions of a derived external class have a range restriction IC then IC is defined for the external class, too. In an object-oriented

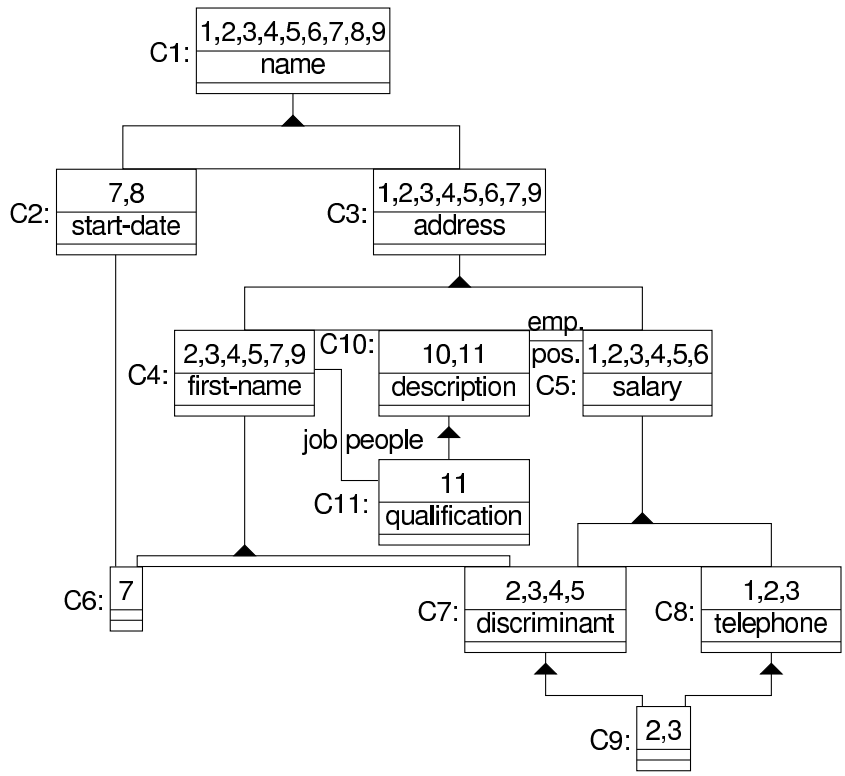


Figure 8.4: Example – first integrated schema

schema, similar to attributes, integrity constraints are inherited. If a subclass has a 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 defined in the external schema. We propose in this case to search the least external class which includes the involved base extensions and to combine the constraint with a discriminant attribute:

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

- *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 have to look for a minimal superclass C containing the base extensions of the constraint on attribute A:

$$\forall o_1, o_2 \in C : o_1(\texttt{discriminant}) \in \{\text{Wert}_1, \ldots, \text{Wert}_n\} \wedge$$

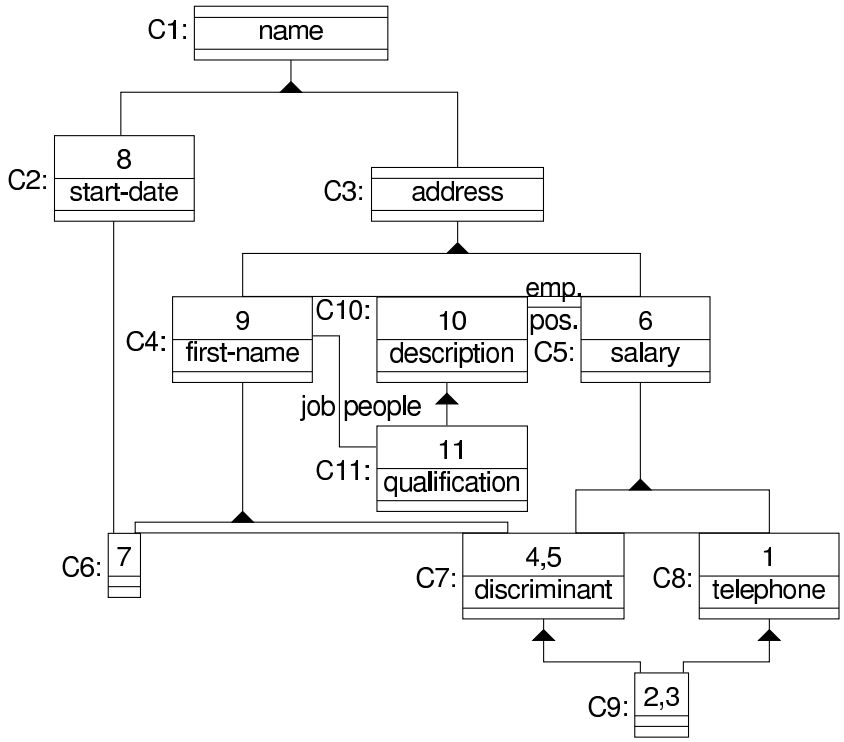


Figure 8.5: Example – reduced integrated schema

$$o_2(\texttt{discriminant}) \in \{\text{Wert}_1, \ldots, \text{Wert}_n\} \Rightarrow$$
$$(o_1(A) = o_2(A) \Rightarrow o_1 = o_2)$$

In our example the integrity constraints of the base extensions are shown in Table 6.6. 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 stated in Table 8.6.

Class	Integrity Constraint
C8	salary ≥ 3500
C7	salary ≥ 3000
C10	description is not null
C10	card(employee) > 0
C10	unique(description)

Table 8.6: Integrity Constraints

The derivation of a normalized, 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 concepts of the discriminant and showed how to deal with integrity constraints. They help to map each object of the external schemata to exactly one base extension.

8.2 Derivation of 'Tailored' Views

In the previous subsection we described the derivation of a normalized, external schema. In correspondence to the 3-level-schema-architecture, cf. [TK78]), external schemata express views for certain applications. One external schema can meet the requirements of a certain application but not the needs of any other application. Usually, the designer of an external schema wants to influence the derivation process corresponding to his certain view. The schema integration methodology must be so powerful that an external schema can be derived which is equivalent to a local schema.

In our GIM approach the designer starts with the normalized, external schema. He is offered many operations to adapt the normalized schema to his view. The operations are presented in Table 8.7.

Category	Operation
translation	complex datatype
	unidirectional reference
homogenization	structure operation
	attribute operation
	renaming
derivation	projection
	class selection
	extensional expansion
	attribute expansion
	removing classes
	merging classes

Table 8.7: View Operations

The first two categories encompass operations introduced in Section 5 and Section 6, respectively. The remaining operations are new operations. 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 Section 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 to unidirectional references. In Section 5 we explained the decomposition of overlapping class extensions into disjoint GIM classes. Our proposed derivation algorithm performs the inverse step.

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 Section 5. The homogenization operations, however, can be performed in both directions. Structure operations, as introduced in Subsection 6.1.1, allow the transformation of attributes into classes and vice versa. Attribute operations, explained in Subsection 6.1.2, encompass composition and decomposition of attributes as well as mapping to new datatypes 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 are often also good external attribute names. An exception is the discriminant which is not available in any local class. 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 information. Thus, the corresponding database mappings cannot be injective mappings.

- *Projection:* Projection means that not each attribute of the normalized schema should be available in the external schema. Therefore, in the integrated schema certain rows are eliminated before our improved derivation algorithm is applied.

  Of course, it is also possible to remove some ticks for certain attributes and base extensions.

- *Class selection:* Not each base extension should appear in an external class. This can be done by eliminating certain columns.

- *Extensional and attribute expansion:* Sometimes the designer wants to derive 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 the 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 rectangle 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 only sometimes improved. Therefore the designer has to decide on the application of these operations. Removing and merging classes are operation defined on the classes of the normalized external schema. The following paragraphs introduce three different 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 classes `C1` and `C3` of Figure 8.5 are abstract classes. An abstract class can be removed without loss of information:

$$Con_{new} = Con \setminus \{C\}$$

Of course, the attributes inherited from the removed superclass must be explicitly made visible in its subclasses and the matrices M and M_N must be computed again. Removing abstract classes reduces the number of external classes but sometimes reduces the understandability of an external schema, too. Therefore, the designer must trade off minimality against understandability.

In our example we remove the classes `C1` and `C3`. Furthermore we give the classes meaningful names and obtain the external schema depicted in Figure 8.6. This external schema is complete because a bijective mapping between the corresponding database states can be defined. Table 8.8 shows the extensional mapping of local classes to external classes and Table 8.9 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.

Removing subclasses: Similar to abstract classes an external schema can contain classes without own attributes. In other words, all attributes of a class $C = (Ext_C, Int_C) \in Con$ 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 belongs to more than one most specialized class. A role concept, cf. [WDS95, GSR96], allows such a multiple membership of objects but is

Global Class	Extension	Local Classes
Trainee (C2)	7,8	Trainee_1
Person-Dep (C4)	2-5,7,9	People_2
Employee (C5)	1-6	Employee_1
Trainee-Cust (C6)	7	$\text{Trainee}_1 \cap \text{Customer}_2$
Employee-Dep (C7)	2-5	Employee_2
Manager (C8)	1-3	Manager_1
Manager-Dep (C9)	2,3	$\text{Manager}_1 \cap \text{Employee}_2$
Position (C10)	10, 11	Position_1
Position-Dep (C11)	11	Position_2

Table 8.8: Example – Global Classes expressed by Local Classes

Local Class	Extension	Global Classes
Person_1	1-8	$\text{Trainee} \cup \text{Employee}$
Employee_1	1-6	Employee
Manager_1	1-3	Manager
Trainee_1	7,8	Trainee
People_2	2-5,7,9	Person-Dep
Customer_2	3,4,7,9	$\text{Person-Dep} \setminus \sigma_{\text{customer = false}}\text{Employee-Dep}$
Employee_2	2-5	Employee-Dep
Cust-Emp_2	3,4	$\sigma_{\text{customer = true}}\text{Employee-Dep}$
Position_1	10,11	Position
Position_2	11	Position-Dep

Table 8.9: Example – Local Classes expressed by Global Classes

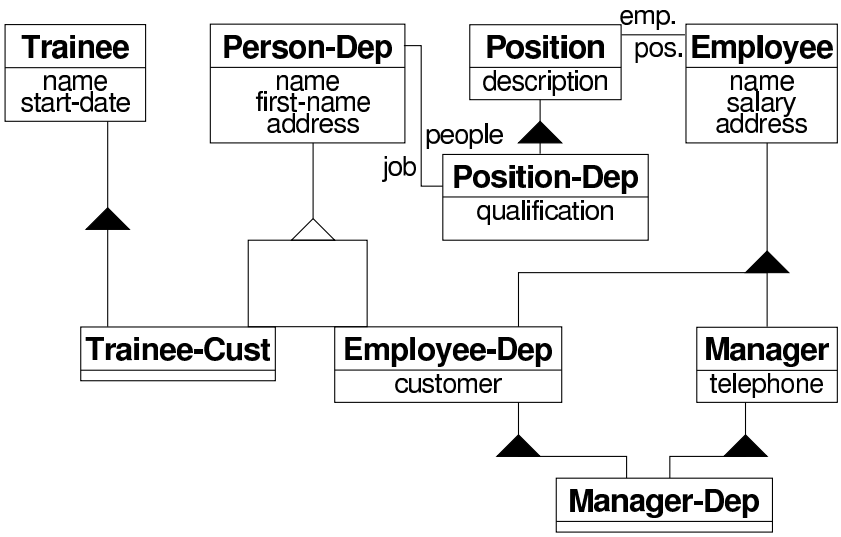


Figure 8.6: Example – final integrated schema

often not implemented in data models of commercial systems. 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 reduced without loss of information 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 the original classes are mutually very similar then null values appear only in few attributes for objects of some base extensions. An extreme situation occurs if you merge all classes into one class which produces a kind of 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, class C_1 and $C_2 \in Con$ have to be merged. Then we obtain the intension and extension of the resulting external class as follows:

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

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

For each attribute from Int and for each base extension from Ext a tick 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 decides to merge the classes `Employee-Dep`, `Manager`, and `Manager-Dep`. The unions of the intensions and extensions are:

$$\begin{aligned} Int &= \{\texttt{name, first-name, address, salary, telephone, pos., job}\} \text{ and} \\ Ext &= \{1, 2, 3, 4, 5\}, \end{aligned}$$

In the integrated schema the designer must set a tick for the base extensions 4 and 5 for attribute `telephone`, and for base extension 1 for the attributes `first-name` and `job`. 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`.

Chapter 9

The Prototype: SIGMA$_{Bench}$

Various database group members of the Otto-von-Guericke-Universität have been working in the project **SIGMA**$_{FDB}$. The topic of this project is schema integration and global integrity maintenance for federated databases. It started in 1995 and is partly funded by the German state Sachsen-Anhalt.

The following three scenarios are applications scenarios for which different tools have been developed. They allow the testing of our research results in order to have a feedback from practical environments.

- *Factory planning:* Our database group cooperated with mechanical engineers from the logistics and factory planning groups of our university and Frauenhofer Institut in Magdeburg. The engineers are using independently developed applications for factory planning. Their problem is the interoperation among their file-based applications, i.e. the usage of the result from one application as the input for another application. The idea was there to demonstrate the advantages of a data federation. We successfully designed an integrated schema and defined mappings to their data files. The prototype **SIGMA**$_{Demo}$ was then developed to demonstrate the way a federated database system works in that scenario.

- *Bio-informatics:* In this scenario many highly heterogeneous databases containing bio-molecular data are available in the Internet. The bio-informatics scientists have been seeing a large success potential if someone integrates these databases. The prototype BioBench is a first prototype to access and to integrate different databases. In this scenario no integrity constraints needed to be considered because the local databases usually offer only read access. The databases are further very often data files, for which schemata had to be extracted before a federation could be established.

- *Global-Info:* This project is a German Digital Library project and is funded by the German Ministry for Education and Research. Different, heterogeneous database systems with publication data exist and have to be integrated. The project started

in 1998 and our group is still working on it. A main problem is to detect same publications in different databases.

For the design of a federated database we developed the tool-box **SIGMA**$_{Bench}$. Main parts of the GIM method are implemented. The architecture of the tool-box is depicted in Figure 9.1. The tools communicate over a central repository managed by the RDBMS

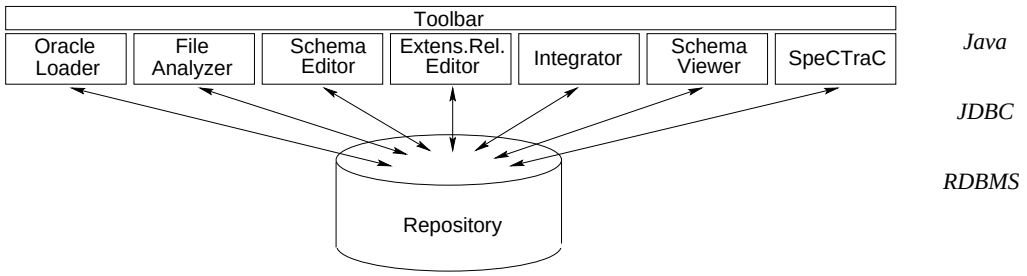


Figure 9.1: Architecture of the **SIGMA**$_{Bench}$

YARD. They are database clients implemented in Java which access the database via JDBC. The following paragraphs introduces the different tools.

Loader: The repository can store database schemata. The loader imports schemata including integrity constraints from an Oracle database and stores them in the repository.

File analyzer: Sometimes there is a need to federate not only databases but files. From files appropriate schemata must be extracted. The file analyzer assists the designer to find key words, tokens, and brackets in files. The schema extraction can be done semi-automatically.

Schema editor and viewer: Besides the import of schemata the designer often wants to create new schemata and to edit existing schemata. The viewer visualize schemata stored in the repository.

Extensional relationship editor: One step during the GIM method is the extensional analysis. The designer can specify extensional assertions in a graphical way. An algorithm computes from the assertions an extension diagram. Some assertions are derived automatically from specialization relationships between classes. Furthermore, this tool validates the consistency of extensional assertions.

Integrator: This tool is the main tool. It generates from given schemata and an extension diagram an integrated schema. It uses our GIM algorithm then to produce a normalized, external schema. The designer can influence the derivation by removing sub- or superclasses. The tool also considers integrity constraints, i.e. it derives constraints on the external schema automatically.

SpeCTraC: This tool assists the specification of global transactions. It supports an extended transaction model where nested transactions and different dependencies among transactions can be defined. The tool analyzes specified transactions and dependencies and check them for consistency.

In this section we only sketched our tool-box **SIGMA**$_{Bench}$. In the appendix we show some screen-shots. For further information look at *http://wwwiti.cs.uni-magdeburg.de/~sigmafdb* and the publication [SST+99].

Chapter 10

Related Work

The problem of schema integration in the context of multidatabases is a relatively old problem. There is a huge number of publications in this area. Most of them were published in the eighties. Various publications use different terms. For example, for the terms 'multidatabase' and 'federated database system' different definitions exist, e.g. [HM85, SL90, LMR90, PBE95, BE96]. We follow the definition of a tightly coupled FDBS published in [SL90]. Integrations problems and solution approaches in loosely coupled FDBS are discussed in [Wie92, ZHK96]. Our schema architecture is similar to the 5-level-schema-architecture from [SL90]. In Chapter 2 we compared our architecture with the famous 5-level-schema-architecture. An interesting modification of this architecture is proposed in [SCRR96] but does not reflect our GIM method.

We did not consider behavior integration. If the behavior is specified in a formal description method, e.g. life cycle diagrams, then results from [SHJ+94, Pre99] can be used for behavior integration.

Sometimes, data files instead of databases need to be federated by an FDBS. The publications [Höd97, SH99] tackle the extraction of schemata and structure information from semi-structured files to support file federation.

Very important for a schema integration is the choice of the right common data model in which the schema integration is performed. The suitability of different data models as common data model is discussed in [BLN86, SCG91, HB96]. The favorite data model is usually an object-oriented data model due to its semantical richness. 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 [SS95, SS96a].

Schema integration means detecting and resolving schema conflicts. In the literature many different conflict classifications were proposed, e.g. in [BL84, KS91, SPD92, SK93, KCGS95, NS96, GSC96]. 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. We have a small and orthogonal set of conflict types. Many conflict types of other publications are combinations of our conflict types.

A hard problem of schema integration is the detection of conflicts. We have assumed 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 [GCS93, GSC95, NS96]. The publications [SK93, KS96] use context knowledge and the idea of semantic similarity to detect correspondences.

The structure conflict is a frequently occurring conflict type. [Mot87, NS96, SPD92], 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 [DH84, LNE89, KCGS95, GSC96]. Following [KCGS95, GSC96] they can be sub-classified in 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. [SPD92] 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 information 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 topic of many publications, e.g. [DH84, Mot87, MNE88, Bra93, SGN93, TS93, KS95, NS96]. They usually resolve this conflict directly in an object-oriented model by using specialization. The original classes are often classes of the integrated schema enriched by new super-/subclasses and specialization relationships among them. [DS96], for example, suggests many operations to resolve a conflict between two classes. Problems arise, however, if two specialization hierarchies with many classes need to be integrated. The mentioned approaches generate in this case very complex schemata. Furthermore, different variants of conflict resolution are often possible. There are no strict rules which help the designer. Therefore, the process of integrating specialization hierarchies is usually very hard for the designer and produces often a huge number of new classes. As mentioned in Section 3 most publications do not correctly analyze extensional relations.

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

The idea of deriving extensional relations from integrity constraints was proposed by [RR97, TS98, TS99].

During the merging of classes integrity constraints must be considered. The publications [BC86, Con86, EJ95, VA96, Ver97] 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 [BC86, Con86, EJ95] handle this conflict as an unsolvable conflict and stop the schema integration. [VA96, Ver97], however, treat conflicting constraints as subjective and ignore them during the schema integration. [RPG95] 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 the fact that a conflict indicates a design failure. Furthermore, the extensional decomposition makes the problem clearer, 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 in a automatic manner. This avoids problems of undecidability which were described in [BC86, Con86, EJ95, EJ96]. Some ideas of our approach were firstly published in [CHS+97, CST97, CST98]. 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 [Tür99].

Objects typically have object identifiers which must be considered during the class merging. [EK91, SS95] introduce approaches to tackling 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 [DeM89, WM89, Ken91, Pu91, KAA+93, LSPR93, ZHKF95, CTK96, WZ96].

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

The derivation of object-oriented, external schemata from a GIM schema is accompanied by a semantical, designer-driven enrichment. [CS91, Cas93, PB94, FV95, Hoh96] propose different enrichment rules defined on relational structures and the concept of primary and foreign keys. Due to the use of GIM we cannot exploit these rules for our approach. In our approach, the improved algorithm itself performs a main part of the enrichment.

In our approach we used a discriminant to avoid loss of information. This problem appears in some publications, e.g. in [KLK91], as conflicting meta information. The resolution of conflicts concerning meta data is topic of [SCG93, CL94]. [CL94] introduces a declarative language to overcome meta conflicts. In our approach we use the idea of discriminants published in [GCS95] and adapted it to our GIM scenario.

There is a huge number of FDBS prototypes and systems. A good overview over the

most important systems is given in [BE96]. In the following we discuss publications of different database groups.

Publications from Navathe and coauthors [NG82, NEL86, MNE88, LNE89, SGN93, NS96]

These papers 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 [MNE88]. Besides this conflict, attribute conflicts and conflicting relationships are considered. They resolve these conflicts similar to the extensional conflict by applying specialization. The combined extension and integrity constraint conflict is topic of [SGN93]. To resolve conflicting integrity constraints [NS96] proposes a logical disjunction for a class union and a logical conjunction for a class intersection. A special case of conflicting constraints is the identification conflict which is discussed by [NG82].

Relationships of two ER-diagrams can stand in conflict, e.g. with respect to arities, roles, and cardinality numbers. [NEL86] resolves such conflicts very similar to the resolution of the extension conflict.

A very good publication is [LNE89] which describes the resolution of extension conflicts, conflicting relationships, and attribute conflicts. [NS96] discusses the problem of detecting conflict correspondences between schemata to be integrated.

Publications from Saltor, Castellanos and Garcia-Solaco [GS91, SCG91, SCG93, GCS93, CSG94, GCS95, GSC95, SCGSK95, GSC96, SCRR96]

The common data model of these publications is the object model BLOOM (Barcelona object-oriented model) introduced in [SCGSK95]. A motivation using an object model is given in [SCG91, CSG94]. The detection of conflict correspondences is shown in [GCS93, GSC95].

For resolution of extensional conflicts the publications [GS91, SCG93, GCS95] propose to generalize conflicting classes. In order to avoid information loss they introduce a discriminant attribute. A 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 [GCS95, Page 26]), especially if an extensional conflict involves many local classes.

Meta conflicts, which are often called *schematic discrepancies* [KLK91], are resolved by using a discriminant [SCG93].

[GSC96] 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 produces wrong integrated schemata.

Publications from Spaccapietra and Dupont [SP91, SPD92, Dup94, DS96]

These publications consider different conflict classes introduced in [SP91]. Conflicts are expressed by assertions. Basing on specified assertions [SPD92] describes different integration rules which resolve these conflicts. This paper considers only identical assertions, i.e. no extensional overlap or inclusion is regarded. Besides extensional conflicts, conflicting binary relationships are resolved by introducing the concept of paths.

Later publications extend the integration method by further extensional conflicts. [DS96] 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.

[Dup94] points out that binary extensional relationships are not sufficient for a correct extensional analysis. He suggests additional types of non-binary extensional assertions. Binary and the new proposed assertions are, however, still incomplete to exactly model extensional relationships among classes.

Publication from Kim and coauthors [KCGS95]

[KCGS95] gives a very detailed classification of conflicts between object-oriented database schemata. The focus there is more on the description of conflicts than on their resolution. For resolving extensional conflicts they describe the operations subclass and union only.

Publications from Ekenberg and Johannesson [Joh93, Joh94, EJ95, EJ96]

In these publications a logic-based approach to schema transformation and integration is used. Before performing an integration the input schemata are transformed into a kind of a normal form, respectively. The common data model has concepts from logic programming and deductive databases. [Joh93] 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. [Joh94] argues that due to the translation of the schemata into the common data model they are getting normalized. Therefore, conflicts can relatively easily be handled.

After translation, two schemata must be checked on conflict-freeness w.r.t. integrity constraints [EJ95, EJ96]. Similar to [BC86, Con86] they point out the problems to detect conflicting integrity constraints. However, they do not take into consideration that conflicting constraints can be caused by wrong designs. For example, local integrity constraints are often too weak or extensional assertions are wrong.

Publication from Motro [Mot87]

[Mot87] is a very early publication about schema integration. It introduces different operations to resolve structure and extensional conflicts. For the structure conflict attributes can be transformed to classes and vice versa. Operations to resolve the extensional conflict are union, intersection, and merging of conflicting classes.

This publication belongs to the first publications which use specialization to resolve extensional conflicts. Furthermore, Motro recognizes the need for an artificial attribute to avoid loss of information.

Publications from Reddy, Prasad and Gupta [RPRG94, RPG95]

The common model of these publications is an object-oriented data model [RPRG94]. Local classes standing in a semantical relationship are represented by a global class. The global class is constructed by merging the local classes. The authors assume identical extensions of related classes. Other extensional conflicts are not regarded.

Attribute conflicts are treated very similar to extensional conflicts. Related attributes are merged into global attributes.

[RPG95] discusses conflicting integrity constraints. The constraints of related classes are disjunctively combined for the global class. Therefore, not each global inserted object can be propagated to local classes.

Chapter 11

Conclusion

In this paper we introduced a new approach to schema integration. In contrast to traditional approaches we distinguish between data model for integration and data model for schemata of global applications. As integration model we use the Generic Integration Model which enables the application of supporting algorithms. The main focus of the GIM method is the consideration of extensional conflicts. We resolve this conflict by decomposing class extensions into disjoint base extensions. As result we are able to use mechanisms of formal concept analysis to derive external schemata.

The GIM method consists of many design steps. Only some parts of a schema integration methodology can be automated. Many work must be done by the designer who should know the semantics of the underlying database systems and applications. We described here design steps in different ways. Informal parts explained the main ideas which are illustrated by appropriate examples whereas formalized parts are used to exactly define the step semantics.

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

Schema integration is in general a very complex task. We do not claim that our approach resolves all kinds of conflicts in the best way. Further research is therefore needed for different aspects, e.g.:

- *reference attributes*: Conflicts between semantically related reference attributes can occur. Resolving this kind of conflicts however requires the idea of specialization of reference attributes which is not supported by most data models.

- *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 enforces to adapt the schemata on top of them and their mappings. A new schema integration is in general too costly. Therefore, there is a need for an incremental schema integration.
- *trade off between correct and fast integration*: The underlying assumption of the GIM method is that the designer prefers correctness rather than integration effort. In many case this assumption is correct, but can produce very large schemata and requires a lot of work for the integration process. For example, the task of extensional reconciliation can be very time-consuming. Some scenarios, however, need a quick and dirty approach. The question is therefore, how can the the GIM approach be modified in that way?

Last but not least, the quality of the GIM approach must be further tested in the praxis.

Bibliography

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

[BE96] O. A. Bukhres and A. K. Elmagarmid, editors. *Object-Oriented Multidatabase Systems — A Solution for Advanced Applications.* Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[BH91] M. W. Bright and A. R. Hurson. Linguistic Support for Semantic Identification and Interpretation in Multidatabases. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems, IMS'91, Kyoto, Japan*, pages 306–313. IEEE Computer Society Press, Los Alamitos, CA, 1991.

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

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

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

[Cas93] M. Castellanos. Semantic Enrichment of Interoperable Databases. In H.-J. Schek, A. P. Sheth, and B. D. Czejdo, editors, *RIDE-IMS'93, Proc. of the 3rd Int. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, Vienna, Austria, April 19–20, 1993*, pages 126–129. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[CHS+97] S. Conrad, M. Höding, G. Saake, I. Schmitt, and C. Türker. Schema Integration with Integrity Constraints. In C. Small, P. Douglas, R. Johnson, P. King, and N. Martin, editors, *Advances in Databases, 15th British*

National Conf. on Databases, BNCOD 15, London, UK, July 1997, Lecture Notes in Computer Science, Vol. 1271, pages 200–214. Springer-Verlag, Berlin, 1997.

[CL93] T. Catarci and M. Lenzerini. Interschema Knowledge in Cooperative Information Systems. In M. Huhns, M. P. Papazoglou, and G. Schlageter, editors, *Proc. of the Int. Conf. Intelligent and Cooperating Information Systems, May 12–14, 1993, Rotterdam, The Netherlands*, pages 55–62. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[CL94] J. Chomicki and W. Litwin. Declarative Definition of Object-Oriented Multidatabase Mappings. In M. T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 375–392. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[Con86] B. Convent. Unsolvable Problems Related to the View Integration Approach. In G. Ausiello and P. Atzeni, editors, *Proc. of the 1st Int. Conf. on Database Theory, ICDT'86, Roma, Italy*, Lecture Notes in Computer Science, Vol. 243, pages 141–156. Springer-Verlag, Berlin, 1986.

[CS91] M. Castellanos and F. Saltor. Semantic Enrichment of Database Schemas: An Object Oriented Approach. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems, IMS'91, Kyoto, Japan*, pages 71–78. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[CSG94] M. Castellanos, F. Saltor, and M. Garcia-Solaco. A Canonical Model for the Interoperability Among Object-Oriented and Relational Databases. In M. T. Özsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*, pages 309–314. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

[CST97] S. Conrad, I. Schmitt, and C. Türker. Dealing With Integrity Constraints During Schema Integration. In *Engineering Federated Database Systems EFDBS'97 — Proc. of the Int. CAiSE'97 Workshop, Barcelona, June 16–17, 1997*, Preprint No. 6, pages 13–22, Fakultät für Informatik, Universität Magdeburg, 1997.

[CST98] S. Conrad, I. Schmitt, and C. Türker. Considering Integrity Constraints During Federated Database Design. In S. M. Embury, N. J. Fiddian, A. W. Gray, and A. C. Jones, editors, *Advances in Databases, 16th British National Conf. on Databases, BNCOD 16, Cardiff, Wales, July 1998*, Lecture Notes in Computer Science, Vol. 1405, pages 119–133. Springer-Verlag, Berlin, 1998.

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

[DeM89] L. DeMichiel. Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains. *IEEE Transactions on Knowledge and Data Engineering*, 1(4):485–493, December 1989.

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

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

[Dup94] Y. Dupont. Resolving Fragmentation Conflicts in Schema Integration. In P. Loucopoulos, editor, *Entity-Relationship Approach — ER'94, Proc. of the 13th Int. Conf. on the Entity-Relationship Approach, Manchester, UK, December 1994*, Lecture Notes in Computer Science, Vol. 881, pages 513–532. Springer-Verlag, Berlin, 1994.

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

[EJ95] L. Ekenberg and P. Johannesson. Conflictfreeness as a Basis for Schema Integration. In S. Bhalla, editor, *Information Systems and Data Management, Proc. of the 6th Conf., CISMOD'95, Bombay, India, November 1995*, Lecture Notes in Computer Science, Vol. 1006, pages 1–13. Springer-Verlag, Berlin, 1995.

[EJ96] L. Ekenberg and P. Johannesson. A Formal Basis for Dynamic Schema Integration. In B. Thalheim, editor, *Conceptual Modelling — ER'96, Proc. of the 15th Int. Conf., Cottbus, Germany, October 1996*, Lecture Notes in Computer Science, Vol. 1157, pages 211–226. Springer-Verlag, Berlin, 1996.

[EK91] F. Eliassen and R. Karlsen. Interoperability and Object Identity. *ACM SIGMOD Record*, 20(4):25–29, December 1991.

[FV95] C. Fahrner and G. Vossen. A Survey of Database Design Transformations Based on the Entity-Relationship Model. *Data & Knowledge Engineering*, 15(3):213–250, 1995.

[GCS93] M. Garcia-Solaco, M. Castellanos, and F. Saltor. Discovering Interdatabase Resemblance of Classes for Interoperable Databases. In H.-J. Schek, A. P. Sheth, and B. D. Czejdo, editors, *RIDE-IMS'93, Proc. of the 3rd Int. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, Vienna, Austria, April 19–20, 1993*, pages 26–33. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[GCS95] M. Garcia-Solaco, M. Castellanos, and F. Saltor. A Semantic-Discriminated Approach to Integration in Federated Databases. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd Int. Conf. on Cooperative Information Systems, CoopIS'95, May 9–12, 1995, Vienna, Austria*, pages 19–31, 1995.

[GS91] M Garcia-Solaco and F. Saltor. Discriminated Operations for Interoperable Databases. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems, IMS'91, Kyoto, Japan.* IEEE Computer Society Press, Los Alamitos, CA, 1991.

[GS98] M. Gertz and I. Schmitt. Data Integration Techniques based on Data Quality Aspects. In I. Schmitt, C. Türker, E. Hildebrandt, and M. Höding, editors, *Proceedings 3. Workshop "Föderierte Datenbanken", Magdeburg, 10./11. Dezember 1998*, pages 1–19, Shaker Verlag, Aachen, 1998.

[GSC95] M. Garcia-Solaco, F. Saltor, and M. Castellanos. A Structure Based Schema Integration Methodology. In P. S. Yu and A. L. P. Chen, editors, *Proc. of the 11th IEEE Int. Conf. on Data Engineering, ICDE'95, Taipei, Taiwan, March 1995*, pages 505–512. IEEE Computer Society Press, Los Alamitos, CA, 1995.

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

[GSR96] G. Gottlob, M. Schrefl, and B. Röck. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems*, 14(3):268–296, July 1996.

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

[HB96] A. R. Hurson and M. W. Bright. Object-Oriented Multidatabase Systems. In O. A. Bukhres and A. K. Elmagarmid, editors, *Object-Oriented Multi-*

database Systems — A Solution for Advanced Applications, chapter 1, pages 1–36, Prentice Hall, Eaglewoods Cliffs, NJ, 1996.

[HM85] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3(3):253–278, July 1985.

[Höd97] M. Höding. Federating Databases and Files: An Approach to a Uniform and Comfortable Interface to Heterogeneous Tourist Information Systems. In A.-M. Tjoa, editor, *Information and Communications Technologies in Tourism 1997, Proc. of the Int. Conf., Edinburgh, Scotland*, pages 140–149. Springer-Verlag, Wien, New York, 1997.

[Hoh96] U. Hohenstein. Using Semantic Enrichment to Provide Interoperability between Relational and ODMG Databases. In J. Fong and B. Siu, editors, *Multimedia, Knowledge-Based & Object-Oriented Databases*, pages 210–232, Springer-Verlag, Berlin, 1996.

[Hul86] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[Joh93] P. Johannesson. Schema Transformations as an Aid in View Integration. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Advanced Information System Engineering, Proc. of the 5th Conf., CAiSE'93, Paris, France, June 1993*, Lecture Notes in Computer Science, Vol. 685, pages 71–92. Springer-Verlag, Berlin, 1993.

[Joh94] P. Johannesson. Using Conceptual Graph Theory to Support Schema Integration. In R. A. Elmasri, V. Kourajian, and B. Thalheim, editors, *Entity-Relationship Approach — ER'93, Proc. of the 12th Int. Conf. on the Entity-Relationship Approach, Arlington, Texas, December 1993*, Lecture Notes in Computer Science, Vol. 823, pages 283–296. Springer-Verlag, Berlin, 1994.

[KAA+93] W. Kent, R. Ahmed, J. Albert, M. Ketabchi, and M.-C. Shan. Object Identification in Multidatabase Systems. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Proc. of the IFIP WG 2.6 Working Conf. on Interoperable Database Systems, DS-5, Victoria, Australia, November, 1992*, pages 313–330. North-Holland, Amsterdam, 1993.

[KCGS95] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. In W. Kim, editor, *Modern Database Systems*, chapter 26, pages 521–550, ACM Press, New York, NJ, 1995.

[Ken91] W. Kent. A Rigorous Model of Object Reference, Identity, and Existence. *Journal of Object-Oriented Programming*, pages 28–36, June 1991.

[KLK91] R. Krishmamurthy, W. Litwin, and W. Kent. Interoperability of Heterogeneous Databases with Schematic Discrepancies. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems, IMS'91, Kyoto, Japan*, pages 144–151. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[KS91] W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer*, 24(12):12–18, December 1991.

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

[KS96] V. Kashyap and A. Sheth. Semantic and Schematic Similarities between Database Objects: A Context-based Approach. *The VLDB Journal*, 5(4):276–304, December 1996.

[LMR90] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, 22(3):267–293, September 1990.

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

[LSPR93] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity Identification in Database Integration. In A. Elmagarmid and E. Neuhold, editors, *Proc. of the 9th IEEE Int. Conf. on Data Engineering, ICDE'93, Vienna, Austria, 19–23 April 1993*, pages 294–301. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[MIR93] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The Use of Information Capacity in Schema Integration and Translation. In S. Baker, R. Agrawal, and D. Bell, editors, *Proc. of the 19th Int. Conf. on Very Large Data Bases, VLDB'93, Dublin, Ireland, August 24–27, 1993*, pages 120–133. Morgan Kaufmann Publishers, San Francisco, CA, 1993.

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

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

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

[NG82] S. B. Navathe and S. G. Gadgil. A Methodology for View Integration in Logical Database Design. In D. MacLeod and Y. Villasenor, editors, *Proc. of the 8th Int. Conf. on Very Large Data Bases, VLDB'82, Mexico City, September 8–10, 1982*, pages 142–164, Morgan Kaufmann Publishers, 1982.

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

[PB94] W. Premerlani and M. R. Blaha. An Approach for Reverse Engineering of Relational Databases. *Communications of the ACM*, 37(5):42–49, May 1994.

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

[Pre99] G. Preuner. *Definition of Behavior in Object-Oriented Databases by View Integration*, Dissertationen zu Datenbanken und Informationssystemen, Vol. 53. infix-Verlag, Sankt Augustin, 1999.

[Pu91] C. Pu. Key Equivalence in Heterogenous Databases. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems, IMS'91, Kyoto, Japan*, pages 314–316. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[RPG95] M. P. Reddy, B. E. Prasad, and A. Gupta. Formulating Global Integrity Constraints during Derivation of Global Schema. *Data & Knowledge Engineering*, 16(3):241–268, July 1995.

[RPRG94] M. P. Reddy, B. E. Prasad, P. G. Reddy, and A. Gupta. A Methodology for Integration of Heterogeneous Databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(6):920–933, December 1994.

[RR97] V. Ramesh and S. Ram. Integrity Constraint Integration in Heterogeneous Databases: An Enhanced Methodology for Schema Integration. *Information Systems*, 22(8):423–446, 1997.

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

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

[SCG93] F. Saltor, M. Castellanos, and M. Garcia-Solaco. Overcoming Schematic Discreprancies in Interoperable Databases. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems, Proc. of the IFIP WG 2.6 Database Semantics Conf., DS-5, Lorne, Victoria, Australia, November, 1992*, pages 191–205. North-Holland, Amsterdam, 1993.

[SCGSK95] F. Saltor, M. Castellanos, M. Garcia-Solaco, and T. Kudrass. Modelling Specialization as BLOOM Semilattices. In H. Kangassalo, H. Jaakkola, S. Ohsuga, and B. Wangler, editors, *Information Modelling and Knowledge Bases VI, Japan*, pages 447–467, IOS Press, Amsterdam, 1995.

[SCRR96] F. Saltor, B. Campderrich, E. Rodriguez, and L. C. Rodriguez. On Schema Levels for Federated DB Systems. In K. Yetongnon and S. Hariri, editors, *Proc. of the 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems, PDCS'96, Dijon, France, September 1996*, pages 766–771. International Society for Computers and Their Application, Six Forks Road, Releigh, NC, 1996.

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

[SH99] K.-U. Sattler and M. Höding. Adapter Generation for Extraction and Querying Data from Web Sources. In *Proc. of 2nd ACM SIGMOD Workshop WebDB'99*, 1999.

[SHJ+94] G. Saake, P. Hartel, R. Jungclaus, R. Wieringa, and R. Feenstra. Inheritance Conditions for Object Life Cycle Diagrams. In U. Lipeck and G. Vossen, editors, *Workshop Formale Grundlagen für den Entwurf von Informationssystemen, Tutzing*, Technischer Bericht No. 3/94, pages 79–89, Universität Hannover, 1994.

[SK93] A. Sheth and V. Kashyap. So Far (Schematically) yet So Near (Semantically). In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems, Proc. of the IFIP WG 2.6 Database Semantics Conf., DS-5, Lorne, Victoria, Australia, November, 1992*, pages 283–312. North-Holland, Amsterdam, 1993.

[SL90] A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SP91] S. Spaccapietra and C. Parent. Conflicts and Correspondence Assertions in Interoperable Databases. *ACM SIGMOD Record*, 20(4):49–54, December 1991.

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

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

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

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

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

[SST+99] K. Schwarz, I. Schmitt, C. Türker, M. Höding, E. Hildebrandt, S. Balko, S. Conrad, and G. Saake. Design Support for Database Federations. In *Proc. Int. Conf. on the Entity-Relationship Approach, ER'99, Paris, November 1999*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1999. *To appear.*

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

[TK78] D. C. Tsichritzis and A. Klug. The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems. *Information Systems*, 3(3):173–191, 1978.

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

[TS98] C. Türker and G. Saake. Deriving Relationships between Integrity Constraints for Schema Comparison. In W. Litwin, T. Morzy, and G. Vossen, editors, *Advances in Databases and Information Systems, Proc. Second East-European Symposium, ADBIS'98, Poznań, Poland, September 1998*, Lecture Notes in Computer Science, Vol. 1475, pages 188–199. Springer-Verlag, Berlin, 1998.

[TS99] C. Türker and G. Saake. Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration. In *Advances in Databases and Information Systems, Proc. Third East-European Symposium, ADBIS'99, Maribor, Slovenia, September 1999*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1999. *To appear.*

[Tür99] C. Türker. *Semantic Integrity Constraints in Federated Database Schemata.* Dissertation, University of Magdeburg, Germany, 1999. *Submitted.*

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II: The New Technologies.* Computer Science Press, Rockville, MD, 1989.

[VA96] M. W. W. Vermeer and P. M. G. Apers. The Role of Integrity Constraints in Database Interoperation. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proc. of the 22nd Int. Conf. on Very Large Data Bases, VLDB'96, Bombay, India, September 3–6, 1996*, pages 425–435. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[Ver97] M. W. W. Vermeer. *Semantic Interoperability for Legacy Databases.* CTIT Ph.D.-thesis No. 97-11. Centre for Telematics and Information Technology, Enschede, The Netherlands, 1997.

[WDS95] R. J. Wieringa, W. De Jonge, and P. A. Spruit. Using Dynamic Classes and Role Classes to Model Object Migration. *Theory and Practice of Object Systems*, 1(1):61–83, 1995.

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

[WM89] Y. Wang and S. Madnick. The Interdatabase Instance Identification Problem in Integrating Autonomous Systems. In *Proc. of the 5th IEEE Int. Conf. on Data Engineering, Los Angeles*, pages 46–55, February 1989.

[WZ96] K. Wang and W. Zhang. Detecting Data Inconsistency for Multidatabases. In K. Yetongnon and S. Hariri, editors, *Proc. of the 9th ISCA Int. Conf. on Parallel and Distributed Computing Systems, PDCS'96, Dijon, France, September 1996*, pages 657–663. International Society for Computers and Their Application, Six Forks Road, Releigh, NC, 1996.

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

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

[ZHKF95] G. Zhou, R. Hull, R. King, and J.-C. Franchitti. Using Object Matching and Materialization to Integrate Heterogeneous Databases. In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proc. of the 3rd Int. Conf. on Cooperative Information Systems, CoopIS'95, May 9–12, 1995, Vienna, Austria*, pages 4–18, 1995.

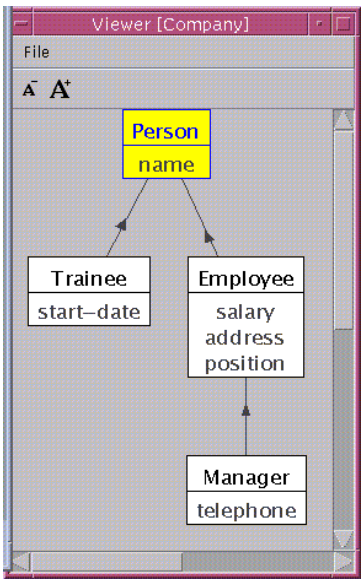


Figure 1: Schema viewer

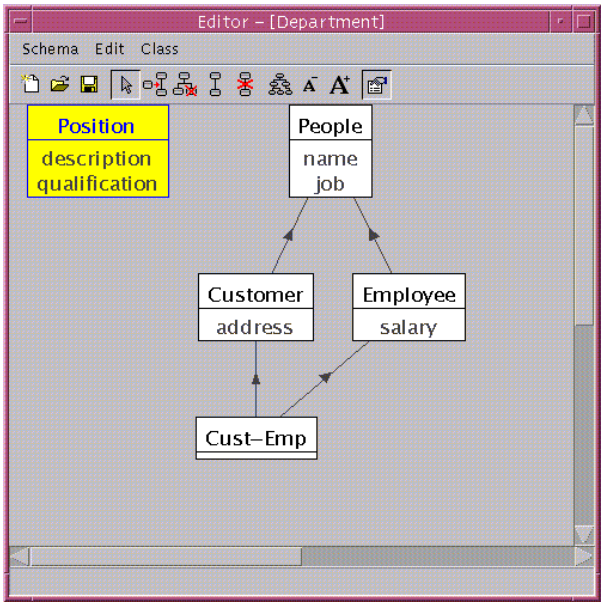


Figure 2: Schema editor

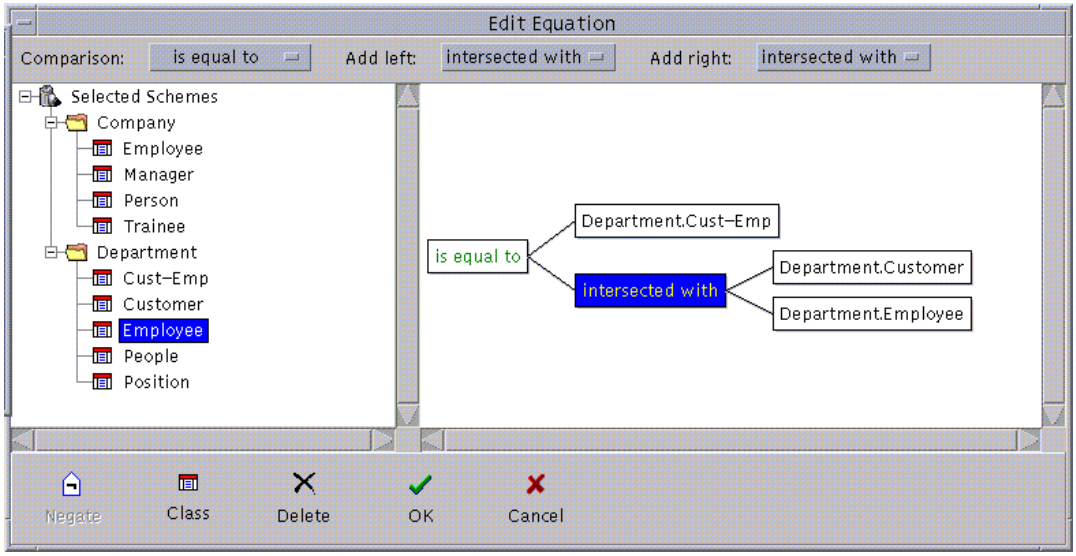


Figure 3: Extensional assertion editor

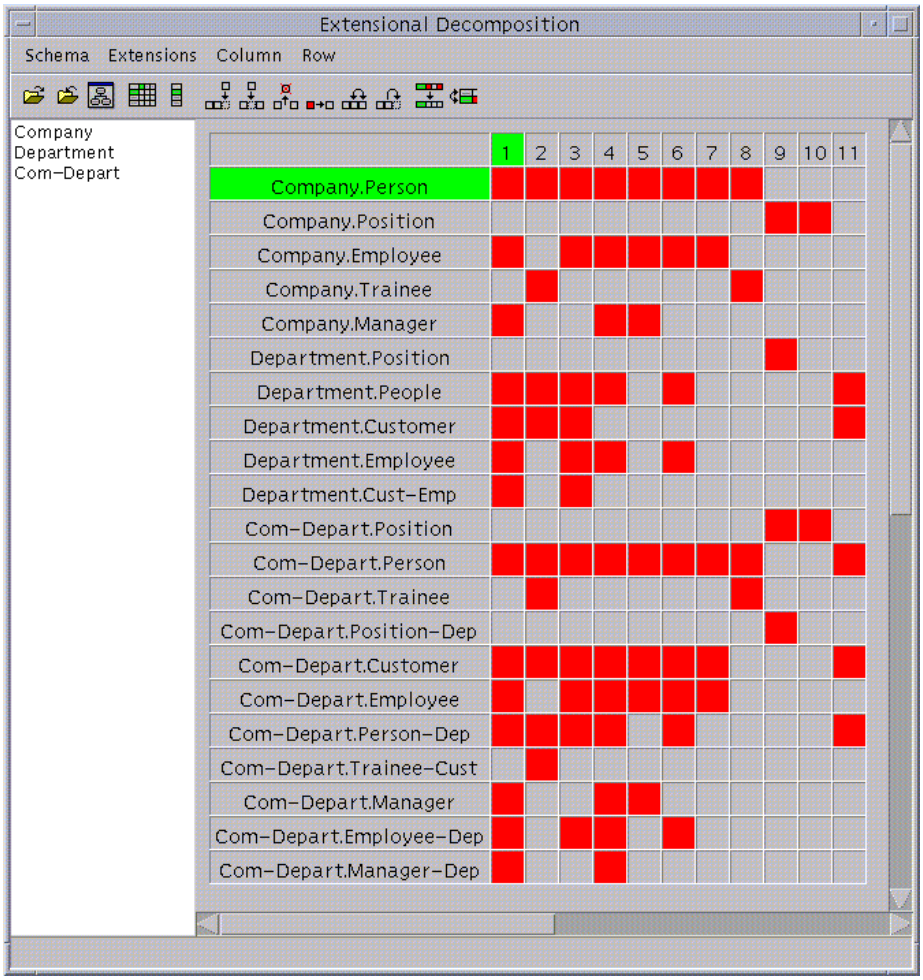


Figure 4: Extension diagram viewer

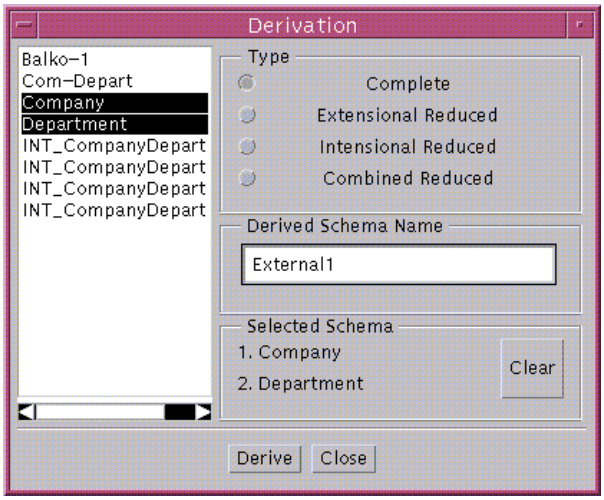


Figure 5: Integrator

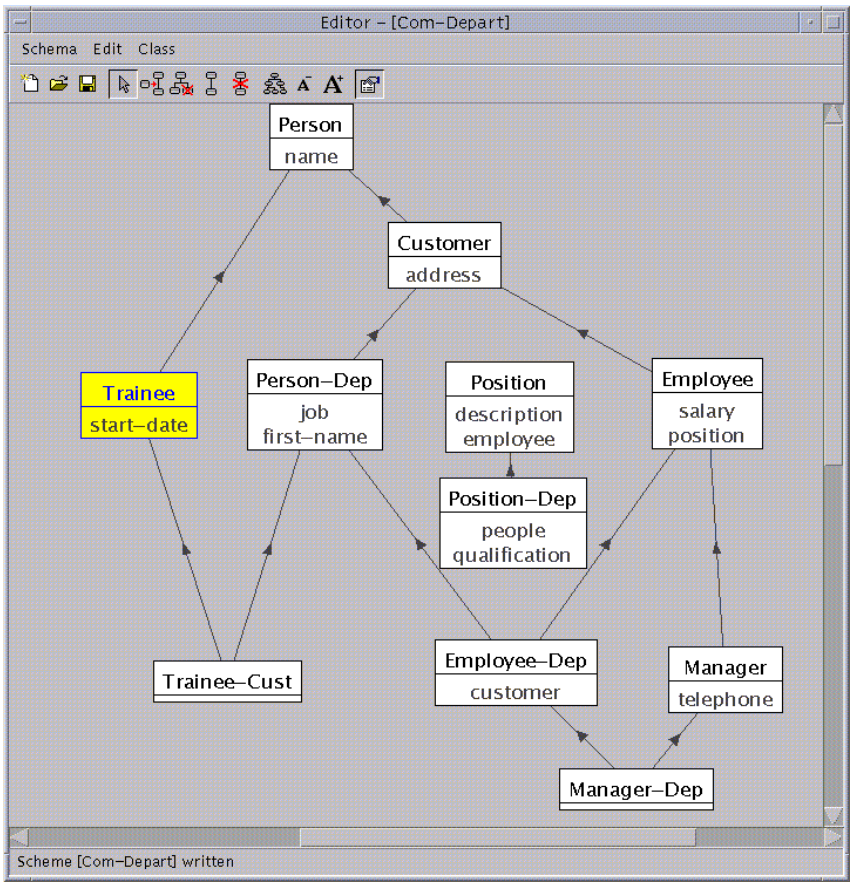


Figure 6: Schema viewer