Appeared as: Preprint Nr. 2, Fakultät für Informatik, Universität Magdeburg, April 1996.

Integrity Constraints in Federated Database Design[1]

Stefan Conrad   Michael Höding   Sven Janssen
Gunter Saake   Ingo Schmitt   Can Türker

Institut für Technische Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Postfach 4120, D–39016 Magdeburg
Tel.: ++49-391-67-18065   Fax: ++49-391-67-12020
E-mail: {conrad|hoeding|janssen|saake|schmitt|tuerker}@iti.cs.uni-magdeburg.de

March 1996

[1]This research was partially supported by the German Country Sachsen-Anhalt under FKZ: 1987A/0025 (Federating Heterogeneous Database Systems and Local Data Management Components for Global Integrity Maintenance) and by the ESPRIT Basic Research Working Group No. 8319 ModelAge (A Common Formal **Model** of Cooperating Intelligent **Age**nts).

Abstract

In this paper we demonstrate the application of a methodology for federated database design. Federated database design includes the transformation of heterogeneous local schemata from the native data models of the component database systems into a common data model, the integration of these homogenized schemata into the federated schema, and the derivation of external schemata for global applications.

Our methodology seamlessly captures all phases from schema transformation and integration up to derivation of external schemata. Our approach differs from other recently proposed methodologies because we consider integrity constraints throughout all phases and use them for resolving conflicts. Therefore, the main focus of this paper is the treatment of integrity constraints and their evolution in the federated database design process.

For resolving conflicts which arise during the transformation and integration process the consideration of integrity constraints can be useful, although the process becomes more complicated. Taking constraints into account provides a better understanding of conflicts. The constraints can be used to judge possible resolutions of conflicts. For certain kinds of conflicts we have to add new global constraints, but often there is not a unique and predictable choice of additional global constraints. The way of resolving such conflicts also depends on the intended semantics of the federated schema and, thereby, on the intended global applications. Sometimes, situations arise where there is even not a consistent solution for all conflicts together. Therefore, a methodology for federated database design must be prepared to make compromises.

Keywords: federated database design, generic integration model, schema transformation, schema integration, integrity constraints.

Contents

1 Introduction **1**

2 Federated Database Systems **3**

3 Classification of Integrity Constraints **7**

4 A Running Example **11**

5 Generic Integration Model **15**

6 Schema Transformation into GIM **19**

7 Schema Integration **25**

8 Derivation of External Schemata **33**

9 Discussion and Outlook **39**

Bibliography **43**

Chapter 1

Introduction

Considering the current situation in enterprises and other organizations we discover that all kinds of database systems are used for different applications in different parts of the enterprise or organization. However, each department (or even worse, each group within a department) usually made its own decision whether to use a database system and then to select a concrete database system. In consequence, there now are different kinds of database systems within an enterprise for which a whole range of application software has been developed independently.

The integration of the different separately existing databases into one logical database is a promising way which can help enterprises to stay competitive while the market is changing faster and faster. Time-consuming and awkward data-transfers between different departments of an enterprise can be avoided. Thus, the cooperation within an enterprise becomes more efficient. For such an integration of generally heterogeneous databases several approaches are possible. The concept of federated databases [HM85, SL90] seems to be most adequate because it does not only provide interoperability between different systems but it also provides a uniform representation of data at the global level. Thereby, a uniform access to heterogeneous databases is possible. Of course, existing applications must be preserved, i.e., we have to take care of that these applications are still applicable on the coresponding local database system. Federated database systems provide a pragmatical way for integrating legacy systems and for gradually replacing legacy systems by modern database systems by migrating applications and data from legacy systems to other database systems (see e.g. [RS94, BS95]).

When integrating already existing databases into a federated database system we have to face a lot of problems: for instance, we have to deal with heterogeneous data models and with heterogeneous database languages. For each of the databases to be integrated there is a conceptual schema and (maybe) additional external schemata for local applications. In order to bring together the different database schemata we have to find a common data model (also called *canonical data model* [SCG91]) and to transform the local conceptual schemata from the local data model into the common data model.

In this paper we present a methodology for federated database design. Our methodology seamlessly captures all phases from schema transformation and integration up to derivation of external schemata. Our approach differs from other recently proposed methodologies (like [SJH93, BFN94, GCS95, RPG95]) due to several reasons. On the one hand our approach is based on a so-called semantically poor data model which allows us to fulfill the requirements of "completeness" and "minimality" put forward by [BLN86]. On the other hand, we consider integrity constraints throughout all phases and use them for resolving conflicts. Therefore, the main focus of this paper is the treatment of integrity constraints in the federated database design process which includes schema transformation, schema integration, and derivation of external schemata.

The remainder of this paper is organized as follows: First, we briefly present the basic notions of federated database systems and consider the process of schema transformation and integration as it is needed for federated databases (Section 2). Then, we introduce some classification characteristics for integrity constraints (Section 3). Based on these characteristics we consider integrity constraints and their evolution during the federated database design process. Section 4 presents an example we are using throughout the paper. In Section 5 we introduce the basics of our *Generic Integration Model* (GIM). Afterwards, we first investigate the schema transformation from the different data models of the component database systems into GIM (Section 6) and then the integration of the resulting GIM schemata into one schema, the federated schema (Section 7). In our presentation, the arising conflicts, their resolution and especially the role of integrity constraints are to the fore. Section 8 shows the derivation of external schemata. In Section 9 we discuss some aspects of our methodology, compare our proposal with related work, and give an outlook on future work.

Chapter 2

Federated Database Systems

In this section we briefly recall the essential notions and issues of federated database systems (for a detailed survey see for instance [SL90, LMR90, BHP92, PBE95]). Due to the fact that there are slightly different understandings of what a federated database system is, we have to clearly state our view which goes along with [SL90].

A *federated database system* (FDBS) is a *distributed* system consisting of a collection of cooperating but (partly) *autonomous* and possibly *heterogeneous* component database systems (CDBSs). The main idea of FDBSs is to provide a *uniform* interface to distributed, heterogeneous data sources of independently developed systems. Thus, new (global) applications have the possibility to access federated data and existing (local) applications operating on the specific CDBSs can continue unchanged. In other words, CDBSs retain their interfaces while they are participating in a federation.

In [SL90] a generally accepted *five-level schema architecture* for federated database systems is introduced:

1. *Local Schema*: A local schema is the conceptual schema of a CDBS which is expressed in the (native) data model of that CDBS.

2. *Component Schema*: A component schema is a local schema transformed into the (common) data model of the federation layer.

3. *Export Schema*: An export schema is derived from a component schema and defines an interface to the local data that is made available to the federation. Thus, only exported schema elements and their data are accessible by the federation layer.

4. *Federated Schema*: A *federated (or integrated) schema* is the result of the integration of multiple export schemata, and thus provides a uniform interface for global applications.

5. *External Schema*: An *external schema* is a specific view on a federated schema and may base on a specific data model different from the common data model. Basically, an external schema serves as a specific interface for global applications.

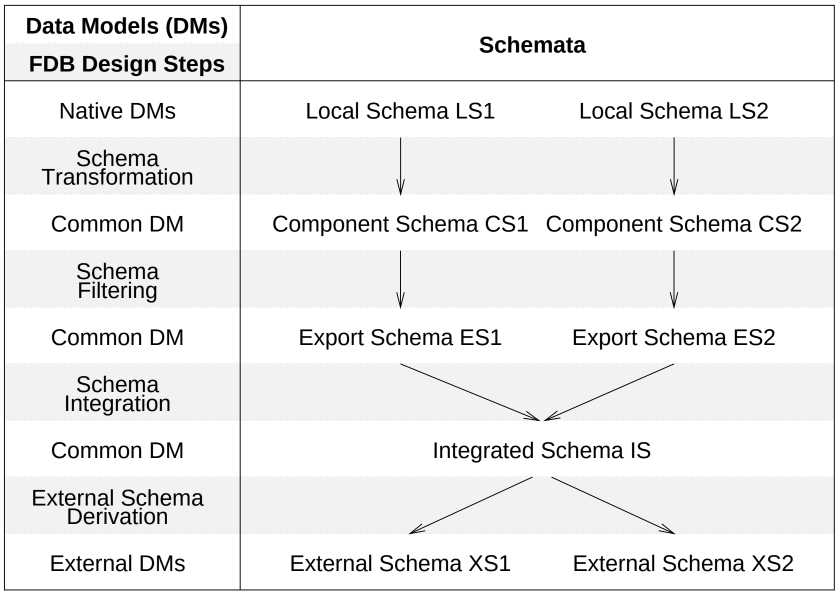

Figure 2.1: Data models, schemata, and steps of the federated database design process

In Figure 2.1 we sketch the common ingredients of the design process of federated databases. A very important issue of the design process of federated databases is the right choice of the common data model. In the literature, see for instance [BLN86, SCG91, PBE95], the proposals for a suitable common data model vary from "semantically rich" data models (e.g. an object-oriented one) to "semantically poor" data models (e.g. the relational model).

The local schemata of the different (heterogeneous) CDBSs represent the starting point for building a federated database. In the first design step the local schemata, which are defined in the data models of the CDBSs, are transformed into the common data model of the FDBS. After this transformation step the data model heterogeneity is overcome. The transformation process is followed by a filtering process. Please note that the order of the transformation and the filtering steps can be changed. The result of these two steps are the export schemata, which may be integrated into a federated schema. The schema integration process itself is very difficult and is subdivided in several steps: preintegration, comparison, conforming, merging, and restructuring of the schemata to be integrated (for details see [BLN86]). After the schema integration the schema heterogeneity (e.g. naming or structural conflicts) is resolved. In the last design step, external schemata which are specific views on a federated schema are derived. In case an external schema should be represented in a data model different from the common data model an additional transformation step is required.

At the end of this section, we focus the attention to the problem of integrity constraint integration. Many existing data models support the (declarative) definition of integrity constraints. Therefore in order to preserve the semantics of a database, integrity constraints have to be transformed, integrated, and derived during the federated database design process, too. As we will see later, an integrity constraint may change its characteristics during the design steps. In order to get a grasp of the treatment of integrity constraints during the design process of federated databases, we first introduce a classification of integrity constraints (see Section 3). On the basis of this classification we later consider the evolution of integrity constraints during the specific steps of the design process.

Chapter 3

Classification of Integrity Constraints

As motivated in the previous section, the integration of integrity constraints plays an important role in the design process of a federated database. There are several kinds of integrity constraints which have to be considered and categorized when dealing with schema and integrity constraints integration [SL90, RPG95]. In this section, we introduce a classification of integrity constraints which enables us later to show how various integrity constraints evolve during the federated database design process.

We classify integrity constraints by two orthogonal dimensions: the *extensional* and the *intensional* dimension. At the extensional dimension, we distinguish between integrity constraints which can be validated on exactly one object[1] and those that must be validated on multiple objects. In our terminology the former ones are called *single object integrity constraints* whereas the latter ones are denoted as *multiple objects integrity constraints*. At the intensional dimension, integrity constraints differ in the number of attributes (zero, one, or more than one) they concern. There are two reasons why we chose this classification. Firstly, this classification corresponds to the concepts of our Integration Model (see Section 5). Secondly, as we will see in the following sections, this classification enables us to clarify how integrity constraints may change their characteristics during the federated database design process. The proposed classification provides the basis for identifying candidates for integrity constraints which may be newly generated during the design process. Additionally, we are able to point out typical evolution pattern of integrity constraints, for instance an integrity constraint originally belonging to only one object class may concern several object classes after the integration process. By combining these two dimensions we obtain six classes of integrity constraints (ICs) (cf. Figure 3.1). Below we give some examples for these classes.

- IC_{10}: These integrity constraints can be validated on exactly one object, whereby

[1] Here, an object refers to exactly one real-world entity. We will discuss this aspect in more detail in Section 5, where we introduce our Integration Model.

Integrity Constraint (IC) Classification		*Extensional Dimension*	
		1 Object	m Objects (m>1)
Intensional Dimension	0 Attribute	IC_{10}	IC_{m0}
	1 Attribute	IC_{11}	IC_{m1}
	n Attributes (n>1)	IC_{1n}	IC_{mn}

Figure 3.1: A classification of integrity constraints

they do not concern any attributes of this object. An example for such kinds of constraints is the inclusion constraint, e.g. an object of a subclass must also be included in its superclass. In this case we can check for each object (instance) of an subclass whether it is also in the extension of its superclass.

- IC_{11}: Examples for this class of constraints are `not null` conditions, restrictions on ranges of attributes, or cardinality constraints on reference attributes.

- IC_{1n}: A typical example for this class are integrity constraints which ensure that certain dependencies between several attributes of an object hold.

- IC_{m0}: Similar to the class IC_{10}, no attributes on object level are required for the validation of such integrity constraints. An example for such kinds of constraints is a condition which is restricting the number of instances a class can have.

- IC_{m1}: The well-known uniqueness condition defined on exactly one attribute of an object (class) is a typical example for such kinds of integrity constraints.

- IC_{mn}: An example for an integrity constraints that must be validated on multiple attributes of multiple objects is the uniqueness condition formulated over a combination of several attributes of an object class.

In the following we discuss several well-known classifications characteristics which may complete our classification.

Intra-Class vs. Inter-Class Integrity Constraints. Often, integrity constraints are divided into *intra-class* and *inter-class* integrity constraints [JQ92]. Whereas intra-class constraints refer to objects of exactly one class, inter-class constraints concern objects of different classes.

Static vs. Dynamic Integrity Constraints. Integrity constraints are usually partitioned into *static* and *dynamic* constraints. Static constraints restrict possible attribute values of objects and thus they restrict possible database states. Dynamic constraints refer to the timely evolution of a database and can be further subdivided into transitional and temporal constraints. Whereas transitional constraints give conditions on possible database state transitions, temporal constraints restrict possible database state sequences to admissible ones [LGS94].

Data Model Inherent vs. Explicit Integrity Constraints. Integrity constraints can also be partitioned into constraints that are *inherent to the data model* and those that must be explicitly formulated.

Descriptive vs. Procedural Integrity Constraints. Integrity constraints differ in the form they are formulated; they may be descriptive or procedural. In the first case the constraints are declaratively written in a higher-level language like SQL. In contrast, procedural constraints are explicitly implemented in application programs or in method bodies (in case of object-oriented systems).

Correlation to Specific Federated Database Design Steps. We especially classify integrity constraints regarding to the specific step of the federated database design process in which they occur (cf. Figure 2.1 in Section 2).

- *Local integrity constraints (LIC)* are the existing constraints of the local databases which are represented in the native data models.
- *Transformed integrity constraints (TIC)* are the local integrity constraints transformed into the common data model.
- *Exported integrity constraints (EIC)* are constraints explicitly defined on the export schema which restrict global access to local data.
- *Integrated integrity constraints (IIC)* are the integrity constraints of the integrated schema derived from the transformed and export integrity constraints.
- *Global integrity constraints (GIC)* are (additional) integrity constraints explicitly defined on the integrated schema.
- *External integrity constraints (XIC)* are constraints explicitly formulated on the external schema to restrict access to the federated database.

Now we can classify integrity constraints w.r.t. the different dimensions: attribute scope, instance scope, class scope, dynamics scope, explicitness degree, and description style. For that, we introduce the following notation which we will use throughout this paper to specify integrity constraints.

<schema><num>$_{<xIC>}$ [<atts>][<exts>] :<condition> (<characteristics>)

The name of the (database) schema an integrity constraint belongs to is given in <schema>. Integrity constraints within a specific schema are distinguished by an additional number <num>. The correlation to a specific federated database design step is expressed by an index (<xIC>) which can be one of the following abbreviations: $LIC, TIC, EIC, IIC, GIC, XIC$ (see above). The set of attributes and the set of extensions referred by the integrity constraint are listed in <atts> and in <exts>, respectively. In <condition> we specify the conditions in a declarative way. Thess conditions can be conditions on values, domains, cardinalities, inclusion dependencies, etc. of the attributes and extensions defined in the square brackets. In <characteristics> we list the (other) classification characteristics, e.g. the class or dynamics scope.

In conclusion, any existing integrity constraint of a local database may evolve throughout the federated database design process w.r.t. all dimensions presented in this section. This paper aims at pointing out principles of this evolution and at showing how this evolution can be managed and used throughout the design process.

Chapter 4

A Running Example

In this section we introduce an example we use in the following sections to illustrate our methodology for federated database design. This example describes two heterogeneous databases of a company. These are a relational database which contains information about the sales department and an object-oriented database which contains information about construction properties of products.

Suppose, the structure of transport machine information (TM) in the sales database is defined in schema A using SQL-DDL:

```
create table transport_machine (
   product_no      char(6) primary key,
   product_name    vchar(40),
   price           number(10,2),
   delivery_period number(3),
   machine_class   vchar(30)) )
```

In this schema only one integrity constraint is explicitly specified. This is the primary key constraint which is intra-class, static and IC_{m1} corresponding to our classification.

$A1_{LIC}$[product_no][TM]: product_no is primary key (IC_{m1}, intra-class, static)

The attributes `price` and `delivery_period` are typical properties of objects in commercial applications. Different machines are classified by the attribute `machine_class`. For this attribute we assume that 'roller conveyor', 'belt conveyor', 'crane' and 'fork-lift truck' are valid strings. But this information is only available in application programs or it is knowledge of the user.

Schema B describes the structure of the construction database according to the ODMG ODL [Cat94]. This database contains information about roller conveyors (RC) and driven roller conveyors (DRC). A important aspect of schema B is the inheritance relationship between roller conveyors and driven roller conveyors.

```
enum project_state_type {in_design, in_construction, completed};
```

```
module construction {
  interface roller_conveyor
  ( extent roller_conveyors
    key product_no )
  {
    attribute string<6>          product_no;
    attribute string<20>         product_name;
    attribute unsigned long      length;
    attribute unsigned long      width;
    attribute project_state_type project_state;

    boolean change(unsigned long length_value, unsigned long width_value);
    boolean set_project_state(project_state_type value);
  };
  ...
    boolean change(unsigned long length_value, unsigned long width_value) {
      if (length_value > 15 * width_value)
        return FALSE; // error
      ...  }
    boolean set_project_state(project_state_type value) {
      if (project_state == completed)
        return FALSE; // error;
      if (project_state == in_construction && value != completed)
        return FALSE; // error
      if (project_state == in_design && value != in_construction)
        return FALSE; // error
      ...}
  ...
  interface driven_roller_conveyor : roller_conveyor
  ( extent driven_roller_conveyors )
  {
    attribute float      max_speed;
    attribute Set<float> motor_voltages;
  }
  ...
}
```

In the following we consider the explicitly specified integrity constraints in schema B in more detail. First, we take a closer look at the key constraint. In schema B the `product_no` is the key of `roller_conveyor`. This integrity constraint is an intra-class, static and IC_{m1} constraint according to our classification.

B1$_{LIC}$[product_no][RC]: product_no is key (IC_{m1}, intra-class, static)

Moreover, there is a constraint defined on the attributes `length` and `width`. We can classify this constraint as IC_{1n}, intra-class, and static.

B2$_{LIC}$[length,width][RC]: length≤15*width (IC_{1n}, intra-class, static)

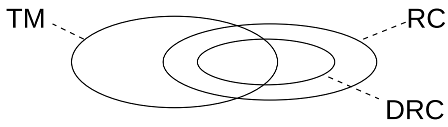

Figure 4.1: Overlapping extensions of example classes

Finally, in schema B there is a dynamic constraint which describes the permitted transitions ($\Rightarrow$) between different project states. This constraint can be classified as intra-class and IC_{11}.

B3$_{LIC}$[project_state][RC]: project_state:in_design$\Rightarrow$in_construction$\Rightarrow$completed
(IC_{11}, intra-class, dynamic)

Now, let us have a look at the extensions of the previously defined classes. For that, we have to describe the meaning of the information stored. The company sales different kinds of transport machines (`TM`). Therefore, information about commercial aspects of the products is stored in the sales database. Another working area of the company is construction and production of roller conveyors (`RC`) which are a special kind of transport machines. Information about the construction of roller conveyors is stored in the construction database. Additionally, the object-oriented concept inheritance is used to express the specialization relationship between a `roller_conveyor` and a `driven_roller_conveyor`. Due to the fact that there are transport machines in the sales database which are also roller conveyors in the construction database the company has to deal with the overlapping extensions of classes in the different databases[1]. However, there are also roller conveyors stored in the construction database which are not contained in the sales database and, vice versa there are transport machines stored in the sales database which are not contained in the construction database. This situation of partially overlapping extensions (cf. Figure 4.1) has to be further considered during the federated database design process.

[1]For this case we assume that objects stored in both databases can be identified by equal product numbers.

Chapter 5

Generic Integration Model

The right choice of the common data model is essential to the design process of an FDBS. This decision influences the quality of the integration step, which is the most difficult step of the design process. Suggested common data models vary from semantically poor data models, e.g. the relational data model, to semantically rich data models, e.g. an object-oriented data model. Most of the recent proposals prefer an object-oriented data model. It is often argued that only a semantically rich data model can be the right common data model because no or only little semantics is lost during the transformation from an object-oriented local schema to the common data model (see also [SCG91]). On the other hand, a semantically rich data model causes more heterogeneity on the schema level than a semantically poor data model does. This heterogeneity on schema level in turn increases the complexity of the integration process. we see this problem in object-oriented integration methodologies as described in [SPD92, RPRG94, PBE95, GSC95]. Therefore, we propose the usage of a semantically poor data model. For best support of the integration process we designed an own data model called the Generic Integration Model (GIM is introduced in [Sch95]). In GIM, only schema information which is relevant for the integration process can be expressed by means of GIM concepts whereas schema information which defines a specific view of an application cannot be expressed. Thus, a schema expressed in GIM offers the flexibility for deriving different external schemata. The view-dependent semantics of a local schema which cannot be expressed in GIM can be reused during the derivation of external schemata. We will show this aspect in the next sections using the example of the previous section. A more detailed discussion about the advantages of using a semantically poor common data model for the integration can be found in [Sch95].

The following list summarizes the basic concepts of GIM:

- *Simple data types*: The concept of a relation in first normal form is adopted from relational database systems in order to define the intension of a GIM class;

- *Non-overlapping class extensions*: The extensions of any two classes have to be either disjoint or identical. Subset relations or partial overlappings between class extensions are not allowed;

- *Object identifier*: In GIM the OID concept is used (see also [KC86]);
- *Bidirectional binary references*: A reference expresses the relationship between exactly two classes. GIM supports different cardinalities (1:1, 1:n, m:n) of reference types between classes. All references in GIM are bidirectional, i.e., each reference has an inverse reference. References are expressed as reference attributes of a class. If a reference attribute can refer to more than one object then the reference attribute is multi-valued. Only reference attributes are allowed to be multi-valued;
- *Integrity constraints*: GIM supports integrity constraints similar to SQL2, e.g. constraints on reference cardinalities, static and dynamic constraints, uniqueness constraints, and constraints which fix an attribute value to all objects of a class.

Due to the semantical poverty of the concepts listed above, GIM cannot function as interface to applications. In consequence, external schemata have to be expressed in another data model than GIM and have to be derived from the integrated GIM-schema. The derivation of an object-oriented external schema from an semantically poor data model (the relational data model) is described in [BKNW91].

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

We present our GIM schema in GIM diagrams (cf. Figure 5.1). In order to understand GIM diagrams we have to explain the semantics of its graphical elements. The horizontal dimension refers to the extensional aspect whereas the vertical dimension refers to the intensional aspect of a class. A class itself is represented by a rectangle. References between two classes are illustrated by lines connecting the respective reference attributes. If integrity constraints are intra-class and single object integrity constraints then they are located within the respective class. Otherwise, the integrity constraints can be found below the diagram. The extensional range of such an integrity constraint is bounded by two arrows. Constraints which fix a specific value to a class attribute (this attribute belongs does not to the intension of the respective class) are located below the diagram. In the Figure 5.1 the integrity constraint `a5=10` is such an integrity constraint on the class attribute `a5`. The names of the extensions and the names of the attributes are located on the top and on the left, respectively, of the diagram.

In the next sections we exemplify the design of a federated database on schema level following the steps described in Section 2. In the example export schemata are not considered.

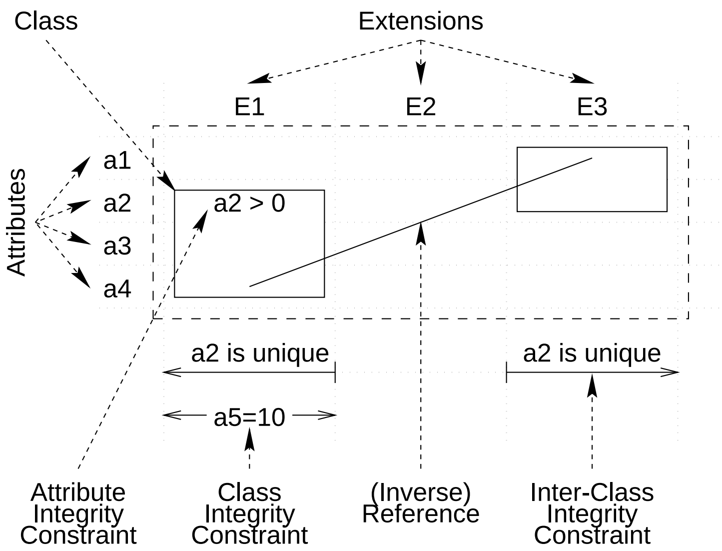

Figure 5.1: Diagram conventions

Chapter 6

Schema Transformation into GIM

As we mentioned in Section 2, the local schemata which are defined in the data models of the CDBS have to be transformed into the common data model GIM. Since the concepts of GIM differ from the concepts of most native data models, these transformations are accompanied by restructuring and normalization steps on the local schemata. The transformation has to be done for each local schema separately. We present the result of the transformation as GIM diagrams. All integrity constraints described in this section are transformed integrity constraints (TIC) and are descriptive. At the component schema level each integrity constraint must have a local counterpart which may be an inherent integrity constraint of the native local data model.

Because of space restriction, not all integrity constraints of our example are described here. We only consider integrity constraints which are interesting in combination with the integration process. The description of the transformation is oriented to the concepts of GIM introduced in the previous section.

Transformation to Simple Data Types

In our example, the first normal form is violated by the definition of attribute `motor_voltages` of class `driven_roller_conveyor`. This attribute has the set-valued data type `Set<float>`. Similar to the normalization process in the relational database design process set-valued attributes have to be transformed into "independent" classes. Therefore, in our example the class `NV`, which represents the class of different motor norm-voltages, is newly generated. The class `NV` has the identifying attribute `voltage`. Two inverse set-valued references between the class `DRC` (abbreviation of `driven_roller_conveyor`) and the class `NV` are generated additionally. The new corresponding reference attributes are the attribute `nv_ref` of the class `DRC` and the attribute `drc_ref` of the class `NV`. The reference to the class `DRC` is only virtual. Its concrete values can be computed by scanning the extension of the local class `driven_roller_conveyor`. The inverse character of both reference attributes is an inherent property of GIM.

In our example the normalization of set-valued attributes causes no change of the explicit local integrity constraints. Instead, three inherent integrity constraints of the local data model become explicit. The first constraint concerns the existence of objects of the class `NV`. Objects of the class `NV` can only exist if they have a reference to at least one object of the class `DRC`:

$B1_{TIC}$[drc_ref][NV]: card(drc_ref)>0 (IC_{11}, intra-class, static)

The next two inherent integrity constraints which become explicit concern the identification of the attribute `voltage`. Due to the value semantics of `NV` objects the attribute `voltage` has to be unique and not null:

$B2_{TIC}$[voltage][NV]: voltage is not null (IC_{11}, intra-class, static)

$B3_{TIC}$[voltage][NV]: voltage is unique (IC_{m1}, intra-class, static)

Now, all non-reference attributes are single-valued. However, the data types of the attributes are not data types of GIM. GIM defines a fixed set of data types differing from the data types used in the local schemata. Therefore, a transformation of the local data types to GIM data types has to be carried out. Problems occur if no isomorphic mapping between the values of a local data type and the values of the GIM data types exists. In this case, explicit integrity constraints can be used to restrict the domain of a GIM data type, e.g. upper bounds, lower bounds, accuracy bounds or enumerated values. Such new explicit integrity constraints are the results of the transformation of integrity constraints which are inherent to the local data types.

Considering our example, we present only two new explicit integrity constraints. Other integrity constraints resulting from the transformation of data types are neglected here. The two integrity constraints restrict the length of string values of the attribute `product_name` of the class `TM` (abbreviation of `transport_machine`) and `RC` (abbreviation of `roller_conveyor`).

$A1_{TIC}$[product_name][TM]: length(product_name)$\leq$40 (IC_{11}, intra-class, static)

$B4_{TIC}$[product_name][RC]: length(product_name)$\leq$20 (IC_{11}, intra-class, static)

Non-overlapping Class Extensions

A very restrictive requirement in GIM is the demand for non-overlapping extensions of GIM classes. This requirement is contradictory to the specialization concept of object-oriented data models. Even in relational schemata this requirement can be violated although no specialization concept exists in the relational data model. Such an violation can be resolved by extensional decomposition. Suppose the extensions of class `A` and class `B` overlap, we generate the classes `A-B`, `B-A` and `A&B` (using the well-known set operations). These three resulting classes are now non-overlapping. The intensions of class `A-B` and class `B-A` are the same as of class `A` and `B`, respectively. The intension of class `A&B` contains all attributes of class `A` and class `B`. In the next sections we show why the demand for non-overlapping class extensions is essential to flexible schema integration and derivation of external schemata.

After the extensional decomposition has been carried out, the relevant local integrity constraints have to be transformed. Single object integrity constraints on a local class can be adopted directly to the classes which are the result of the extensional decomposition of the local class. In general, this procedure cannot be applied to multiple objects integrity constraints. Multiple objects integrity constraints have to be checked against a specific set of objects. Therefore, if such an integrity constraint is an intra-class constraint and the specific set of objects is split by an extensional decomposition, this integrity constraint becomes an inter-class constraint defined on the union of the resulting class extensions.

In our example, the extensions of the classes `roller_conveyor` and `driven_roller_conveyor` overlap. By extensional decomposition we obtain the classes `DRC` and `RCnotDRC`. The class `RCnotDRC` contains objects which are instances of the local class `roller_conveyor` but are not instances of the class `driven_roller_conveyor`. The intension of class `RCnotDRC` is adopted from the class `roller_conveyor` whereas the intension of the class `DRC` contains the attributes from both classes `roller_conveyor` and `driven_roller_conveyor`.

Now we have to transform the explicit integrity constraints defined on roller conveyors. The transformation of single object integrity constraints to the classes `RCnotDRC` and `DRC` will be shown later. In the local schema there exists a multiple objects integrity constraint on the class `roller_conveyor`, namely $B1_{LIC}$, which specifies the key property for the attribute `product_no`. The key property on the attribute `product_no` defined in the ODMG ODL corresponds to a uniqueness integrity constraint on the respective attribute. This integrity constraint is an intra-class and multiple objects integrity constraint and becomes an inter-class integrity constraint after the decomposition of class `RC`. The resulting integrity constraint has to be checked against the union of the extensions of classes `RCnotDRC` and `DRC`.

$B5_{TIC}$[product_no][RCnotDRC,DRC]: (RCnotDRC $\cup$ DRC).product_no is unique (IC_{n1}, inter-class, static)

Since the attributes of class `roller_conveyor` are now attributes of the classes `RCnotDRC` and `DRC`, too, the single object integrity constraint $B4_{TIC}$ concerning the attribute `product_name` has to be adopted.

$B4_{TIC}$[product_name][RCnotDRC]: length(product_name)$\leq$20 (IC_{11}, intra-class, static)

$B4_{TIC}$[product_name][DRC]: length(product_name)$\leq$20 (IC_{11}, intra-class, static)

Object Identifiers and References

GIM supports the object identifier concept in order to identify objects. The identification of local objects, especially in relational database systems, can differ from the

GIM concept. In [SS95, EK91, Ken91] different ways to resolve identification conflicts in FDBS are shown.

Integrity Constraints

All relevant integrity constraints which are not dealt with up to now, are transformed as follows:

- In the local construction schema a procedural transitional integrity constraint is defined on the attribute `project_state` of the class `roller_conveyor`. Due to the fact that GIM does not support procedural integrity constraints, this integrity constraint has to be transformed to an explicit and declarative constraint on the component schema level. In general, this transformation cannot be done automatically because the examination of implementation code is a typical undecidable problem. Thus, human assistance is inevitable. After the transformation has been done, the resulting explicit transitional integrity constraint has to be restructured in correspondence with extensional decomposition of class `RC`. This integrity constraint ($B3_{LIC}$) is a single object local integrity constraint. Therefore, it can be adopted to the classes `RCnotDRC` and `DRC`, and remains an intra-class integrity constraint.

  $B6_{TIC}$[project_state][RCnotDRC]: project_state: in_design⇒in_construction⇒completed (IC_{11}, intra-class, transitional)

  $B6_{TIC}$[project_state][DRC]: project_state: in_design⇒in_construction⇒completed (IC_{11}, intra-class, transitional)

- Another procedural single object integrity constraint is implemented in the method `change` of class `roller_conveyor` of the construction schema ($B2_{LIC}$). Similar to the previous integrity constraint, it has to be transformed to an explicit integrity constraint, and it can be adopted to the classes `RCnotDRC` and `DRC` and remains an intra-class integrity constraint.

  $B7_{TIC}$[length,width][RCnotDRC]: length>15*width (IC_{1n}, intra-class, static)

  $B7_{TIC}$[length,width][DRC]: length>15*width (IC_{1n}, intra-class, static)

- In the relational schema of the sales database the attribute `product_no` of the relation `transport_machine` was defined as primary key ($B1_{LIC}$). In the relational model, a primary key attribute is always unique and not null for all tuples of the respective relation. This integrity constraint can be decomposed into two integrity constraints and can be directly adopted to the class `TM`.

  $A2_{TIC}$[product_no][TM]: product_no is unique (IC_{m1}, intra-class, static)

  $A3_{TIC}$[product_no][TM]: product_no is not null (IC_{11}, intra-class, static)

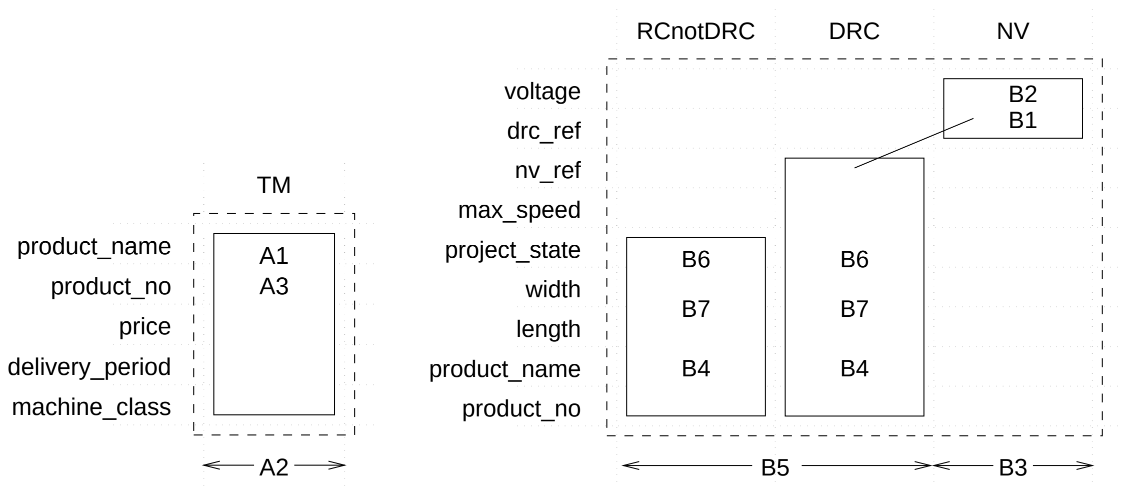

Figure 6.1: Schema A and B transformed to GIM

The component schemata are depicted in Figure 6.1 as GIM diagrams.

Chapter 7

Schema Integration

After all local schemata have been transformed into the common data model the data model heterogeneity is resolved. Now, in order to overcome the schema heterogeneity the component schemata have to be integrated. This integration is a very complex task. Many kinds of conflicts between two schemata can occur and have to be resolved. A prerequisite for conflict resolution is that all conflicts can be detected. Such a detection cannot be done automatically but hat be assisted by an integration tool. A methodology which considers different kinds of conflicts and describes steps for their resolution. In this section, we sketch our schema integration methodology examplarily using the running example. Because of space restrictions, we cannot discuss all possible kinds of conflicts that may occur during the integration process, e.g. the naming conflict [BH91] and further conflicts as in [KCGS95]. The following discussion is therefore restricted to the conflicts which occur in our example:

1. Meta conflicts
2. Attribute conflicts
3. Intensional conflicts
4. Extensional conflicts

In order to improve the comprehensibility of the complex integration we show the results of the homogenizing operations in several GIM diagrams. Furthermore, we focus on the aspect how integrity constraints have to be dealt with during the integration steps.

Resolving Meta Conflicts

The meta conflict exists in the case, when objects of one class differ in values of a specific attribute and corresponding objects of another schema differ in their class assignment. The attribute values of the first schema correspond to classes of the other schema. This conflict is referred in [KLK91] as *schematic discrepancies*. The meta conflict is further described in [CL94]. In our example the value "roller conveyor" of the attribute `machine_class` of class `TM` corresponds to the classes `RCnotDRC` and `DRC`. In other words, transport machines of which the attribute `machine_class` is set to "roller conveyor" correspond to objects of the classes `RCnotDRC` and `DRC`.

Meta conflicts can be resolved by the transformation of the class which presents the attribute variant to the corresponding class variants. In our example the class `TM` has to be split into the classes `TMnotRC` and `TMisRC`. The class `TMnotRC` contains all objects of which the value of the attribute `machine_class` is different to "roller conveyor" whereas the class `TMisRC` contains only objects of which the value of the attribute `machine_class` is "roller conveyor". The intensions of both resulting classes are adopted from the class `TM`. This decomposition operation transforms two data model inherent integrity constraints into two explicit integrity constraints, which restrict the values of the attribute `machine_class`. Since the value of the attribute `machine_class` of class `TMisRC` is restricted to the fixed value "roller conveyor" this attribute can be dropped from the intension and this restriction can be considered as a class attribute with a fixed value. The two resulting integrity constraints are:

$A4_{IIC}$[machine_class][TMisRC]: machine_class='roller conveyor' (IC_{10}, intra-class, static)

$A5_{IIC}$[machine_class][TMnotRC]: machine_class≠'roller conveyor' (IC_{11}, intra-class, static)

Furthermore, two single object integrity constraints ($A1_{TIC}$ and $A3_{TIC}$) are adopted to the classes `TMisRC` and `TMnotRC`.

$A1_{IIC}$[product_name][TMisRC]:length(product_name)≤40 (IC_{11}, intra-class, static)

$A1_{IIC}$[product_name][TMnotRC]:length(product_name)≤40 (IC_{11}, intra-class, static)

$A3_{IIC}$[product_no][TMnotRC]: product_no is not null (IC_{11}, intra-class, static)

$A3_{IIC}$[product_no][TMisRC]: product_no is not null (IC_{11}, intra-class, static)

The multiple objects constraint $A2_{TIC}$ cannot be adopted directly to the classes `TMisRC` and `TMnotRC`. This uniqueness constraint becomes an inter-class constraint and is defined on the union of the extensions of class `TMisRC` and `TMnotRC`.

$A2_{IIC}$[product_no][TMisRC,TMnotRC]: (TMisRC ∪ TMnotRC).product_no is unique (IC_{m1}, inter-class, static)

The resulting GIM classes of the construction data base are depicted in Figure 7.1.

Resolving Attribute Conflicts

After the resolution of the meta conflict the intensions and the extensions of the transformed local class are fixed. In order to merge two schemata into one schema (GIM diagram) and in order to locate the classes to each other in the intensional dimension,

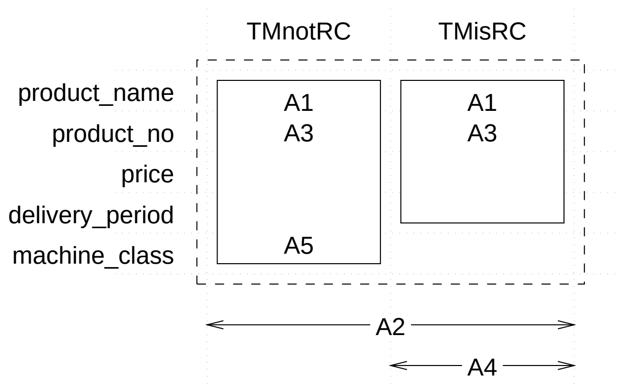

Figure 7.1: GIM schema A after resolving the meta conflict

the attributes of the two schemata have to be compared to each other. Two or more attributes are considered as one attribute in the integrated schema if they are semantically equivalent. The semantical equivalence of attributes has to be established by the integration administrator. Attribute conflicts occur if the domains of several attributes, which have been declared as semantically equivalent before, are not compatible (see also [DeM89]). Such conflicts can be resolved by defining value pairs in form of lookup tables or procedural functions in case there is an isomorphic mapping between the corresponding domains. If no isomorphic mapping can be found then splitting of the attributes or generating of additional integrity constraints can be used to resolve this conflict. Due to space restrictions a detailed description of attribute conflicts cannot be discussed here.

In our example all attributes which have same names are semantically equivalent and are compatible. One exception concerns the attributes `product_name`. Since the maximal string lengths of these attributes are restricted to 40 in schema A and to 20 in schema B they are not compatible and no isomorphic mapping between them can be found. In this case, the heterogeneity is expressed by different integrity constraints which are defined on different extensions. This conflict will be further dealt with during the resolution of extensional conflicts and during the derivation of external schemata in Section 8. The integrity constraints which concern the attribute `product_name` are:

$A1_{IIC}$[product_name][TMisRC]:length(product_name)$\leq$40 (IC_{11}, intra-class, static)

$A1_{IIC}$[product_name][TMnotRC]:length(product_name)$\leq$40 (IC_{11}, intra-class, static)

$B4_{IIC}$[product_name][RCnotDRC]:length(product_name)$\leq$20 (IC_{11}, intra-class, static)

$B4_{IIC}$[product_name][DRC]:length(product_name)$\leq$20 (IC_{11}, intra-class, static)

Now the classes of both schemata can be arranged vertically (intensionally) to each other. An open question concerns the horizontally (extensionally) arrangement of the classes. From the extensional relationships between the local classes (cf. Figure 4.1) and the restruction operations up to now, the extensional relationships between the restructured classes can be derived automatically. The result is depicted in Figure 7.2.

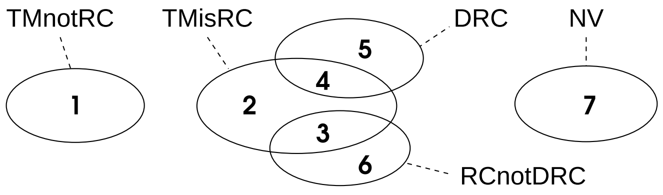

Figure 7.2: Class extensions on integrated schema level

We have now reached the point where the classes of both schemata can be put into one schema (cf. Figure 7.3). In order to illustrate all extensional overlappings the extension of class `RCnotDRC` is located left and right of the extension of class `DRC` in Figure 7.3. This graphical decomposition must not be considered as a restructuring of the schema. It is necessary only for illustration purposes.

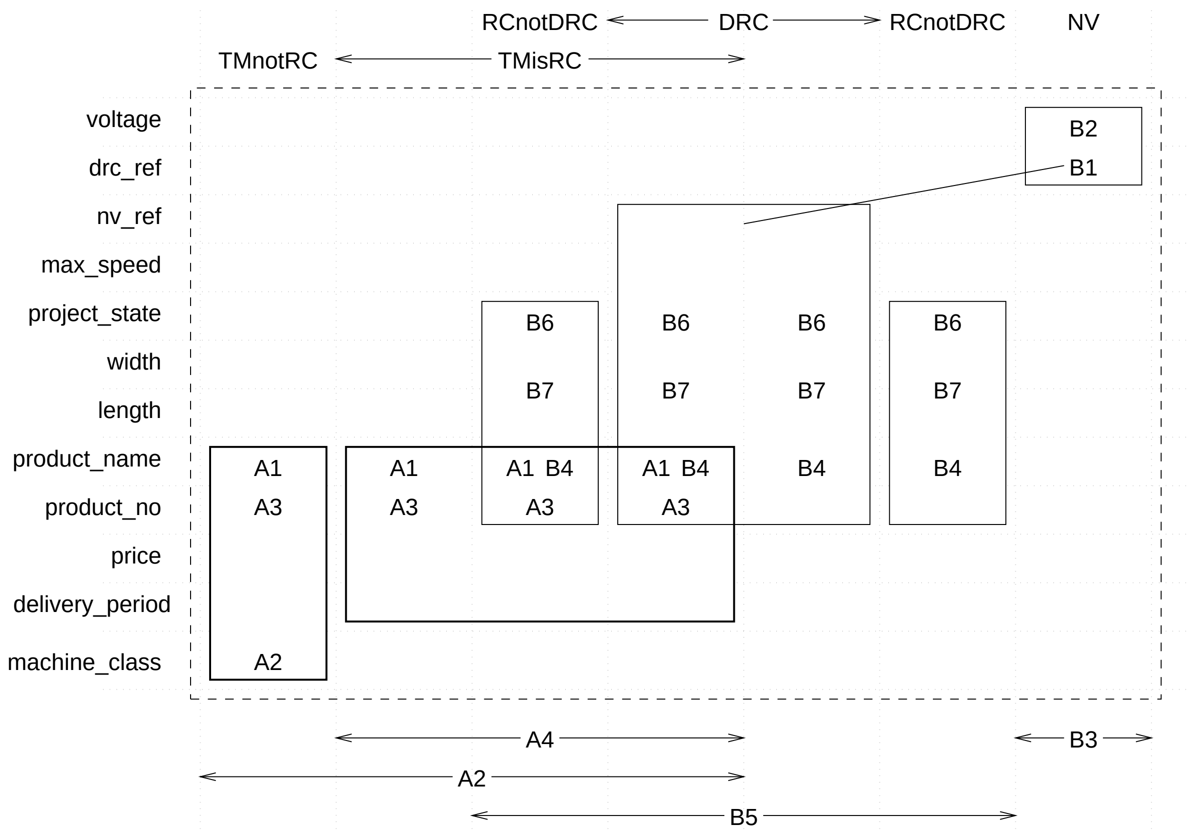

Figure 7.3: GIM classes in one diagram

As we can see in Figure 7.3, the intensions and extensions of the classes overlap. However, an extensional overlapping is not allowed in GIM. In the next subsections we show how intensional and extensional overlappings are overcome.

Resolving Intensional Conflicts

The intensional overlapping hinders the derivation of external schemata. We solve this conflict by intensional decomposition. Intensional decomposition means to group attributes in such way that all attributes of a group are attributes of the same set of classes. Following this criterion equivalence classes on the set of all attributes are established. Corresponding to the equivalence classes the GIM classes can be decomposed intensionally. Intensional decompositions are represented graphically by horizontal lines. If a class is decomposed intensionally then the extensions of the resulting classes are the same but their intensions are disjoint. The intensional decomposition does not influence integrity constraints because they are defined on class extensions. One exception occurs if an integrity constraint is defined on more than one attribute of a class and these attributes are separated by the intensional decomposition. In this case an intra-class integrity constraint becomes an inter-class integrity constraint.

Resolving Extensional Conflicts

The resolution of extensional conflicts is more complex than the resolution of intensional conflicts due to the different consequences for the integrity constraints. In order to meet the demand for disjoint or identical class extensions in GIM we have to decompose extensionally overlapping classes. Suppose, class `A` and class `B` overlap extensionally. The result of the decomposition of the classes `A` and `B` are the three classes `A-B`, `B-A` and `A&B` of which the extensions are disjoint. Objects of the class `A&B` can be detected as described in [LSPR93, Pu91, ZHKF95]. Because intensional conflicts are resolved the intensions of `A` and `B` are the same and are adopted to the classes `A-B`, `B-A` and `A&B`. The decomposition of classes can cause changes of integrity constraints. This aspect is already described in the subsection 'Non-overlapping class extensions' of Section 6. As a result of extensional decomposition the class `A&B` is generated. In general, there are different integrity constraints defined on the original classes. These integrity constraints have to be combined and assigned to the global class `A&B`. Due to the fact that the original class extensions are valid in correspondence with their integrity constraints and the extensions refer to the same set of real-world entities, we can combine the integrity constraints conjunctively to global integrity constraints.

The class `A&B` hides redundancy. All information represented by this class is stored in both databases. In general, in both databases the attribute values of the objects are not equal. The problem is how to deal with such inconsistencies. One way is the generation of new integrity constraints which requires equivalence of all attribute values of such objects. Such integrity constraints have to be hidden from global applications. In order to guarantee consistency with regard to this integrity constraint, the autonomy of the component database management systems has to be restricted. If local applications change the state of one database then this change has to be propagated to another database management system which manages information about the same real-world entities. Furthermore, before global applications are allowed to read federated data, all

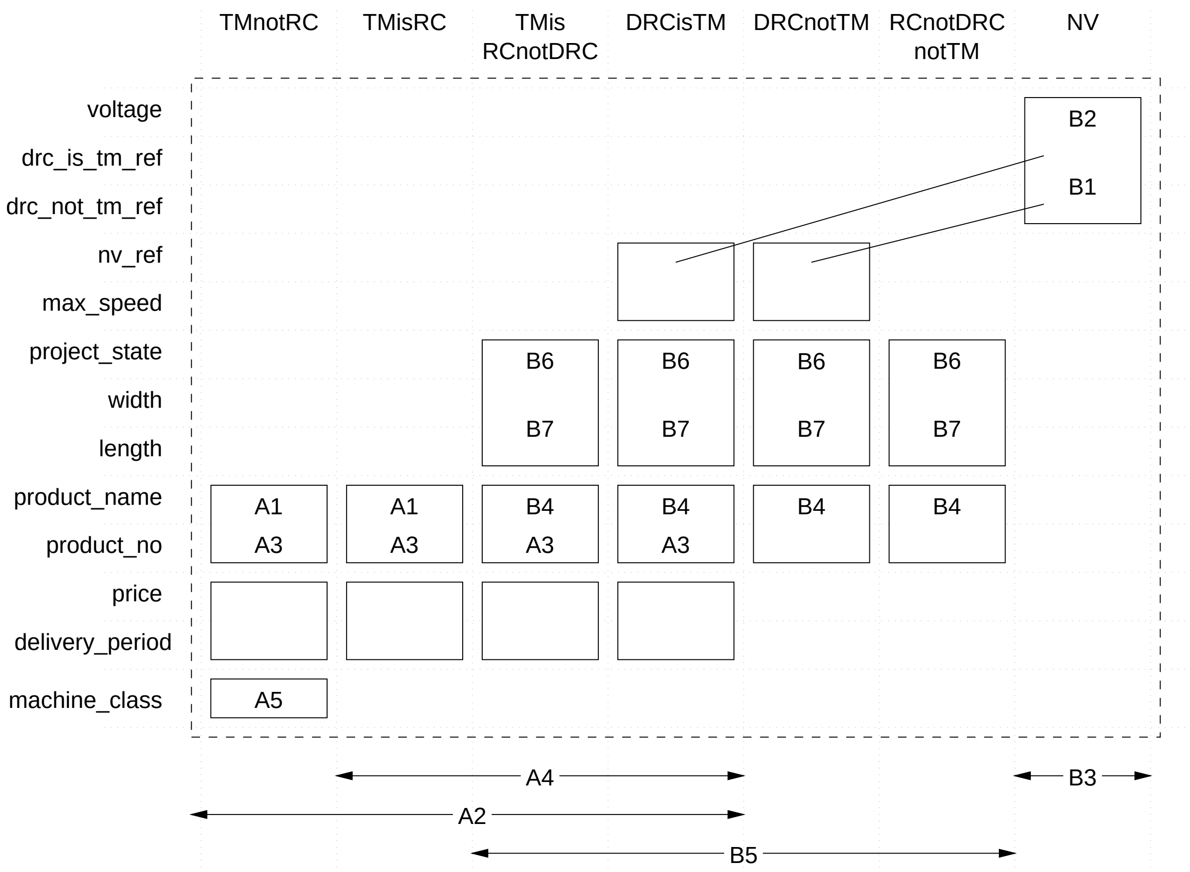

Figure 7.4: GIM schema after intensional and extensional decomposition

inconsistent data has to be made consistent before. In other words the data heterogeneity has to be overcome.

If we apply the extensional decomposition to our example then we obtain the integrated schema as it is depicted in Figure 7.4.

The resulting class extensions have to be given names. In general, this is a very difficult task. Therefore, we construct the names by means of set operations ('is' refers to intersection and 'not' refers to difference).

After the extensional decomposition the resulting schema can be inconsistent with respect to the GIM property of bidirectional binary references. In our example, the class `DRC` was split into the classes `DRCisTM` and `DRCnotTM`. Both resulting classes have a reference attribute to the class `NV`. The question is now which class is the target class of the inverse reference of class `NV`. The solution of this conflict is the splitting of the attribute `drc_ref` of class `NV` into the reference attributes `drc_is_tm_ref` and `drc_not_tm_ref` which are directed to class `DRCisTM` and class `DRCnotTM`, respectively. The splitting of the reference attribute requires an adjustment of the integrity constraint $B1_{TIC}$.

> $B1_{IIC}$[drc_is_tm_ref,drc_not_tm_ref][NV]: card(drc_is_tm_ref)+card(drc_not_tm_ref)>0
> (IC_{1n}, intra-class, static)

In our example the extensional decomposition of the class `TMisRC` and class `DRC` generates the class `DRCisTM`. This global class contains objects which are objects of both databases simultaneously and refer to the same real-world entity (if they have same values of the attribute `product_no`). The local integrity constraints have to be combined conjunctively and have to be adopted to the class `DRCisTM`. The local integrity constraint $B4_{TIC}$ is more restrictive than integrity constraint $A1_{TIC}$. Due to the conjunctive combination of these integrity constraints $A1_{TIC}$ becomes useless and can be dropped.

Global Integrity Constraints

Global integrity constraints are additional integrity constraints defined on the integrated schema. They can be used to enrich the integrated schema which then can serve as basis schema for deriving external schemata. The enrichment of an integrated schema by additional integrity constraints is useful to reduce the conflict potential arising during the process of deriving external schemata (see the next section). However, additional integrity constraints restrict availability of federated data and make the derivation of an external schema which is equivalent to a local schema more difficult.

In our example, the integrity constraints $A2_{IIC}$ and $B5_{IIC}$ are defined on the same attribute `product_no` but not on the same extensions. This conflict can be resolved by adding the following global integrity constraint, which is stronger than $A2_{IIC}$ and $B5_{IIC}$:

> $G1_{GIC}$[product_no][TMnotRC, TMisRC, TMisRCnotDRC, DRCisTM, DRCnotTM, RCnotDRCnotTM]: (TMnotRC $\cup$ TMisRC $\cup \ldots \cup$ RCnotDRCnotTM).product_no is unique
>
> (IC_{m1}, inter-class, static)

When we add this global integrity constraint to the integrated schema, the existing local databases can violate this constraint. Therefore, the federated database may have to be updated. The problem now is to decide which objects of the local databases are valid and which are not. In general, this decision cannot be done automatically. Furthermore, updating the local databases can violate the demand for autonomy of the component database systems.

Chapter 8

Derivation of External Schemata

In this section we outline the derivation of external schemata. First, we discuss the motivation of the derivation process. After this some basic ideas behind our derivation methodology will be explained. In the following we demonstrate the derivation of an external schema using our example. Especially the evolution of integrity constraints is discussed in more detail.

The derivation of external schemata is another very important issue of the federated database design process. Due to the fact that global applications need specific views on the federated data, the derivation of different external schemata has to supported. The external schema can base on the common data model or another data model. However, as mentioned before, we designed GIM only as an integration data model and that is why it should not be used as data model of an external schema. Moreover, we define three kinds of external schemata:

1. An external schema has to equal a local schema. This case is a prerequisite for the migration of local applications onto the top of the federated system. Such a view can provide more data for the migrated application and enables database system independence.
2. Sometimes a complete view on the entire federated database is needed. Therefore, an external schema has to be derived which includes all classes of the integrated schema.
3. For some applications restricted views on the federated schema are required.

In order to derive these kinds of external schemata different transformation steps have to be carried out.

1. The derivation of an external schema which is equal to a local schema requires the inverse execution of transformation and integration operations applied to the integrated schema. Therefore, our methodology supports operations of deriving external schemata which are inverse to the homogenizing operations. Please note, that

such an external schema cannot be derived correctly from the integrated schema if the integrated schema is enriched by additional global integrity constraints.

2. The integrated schema itself presents a complete view on the entire federated database. Due to the fact that GIM should not function as an external data model, the integrated schema has to be transformed to another data model. Furthermore, for a better comprehensibility the classes of the integrated schema can be composed to greater ones without loss of information.

3. A restricted view can be derived by dropping schema elements and by adding new integrity constraints similar to select conditions for views in relational database systems.

Besides the listed tasks, additional restructuring operations can be applied to generate differently structured views. In the following, we focus on the derivation of complete external schemata based on an object-oriented data model. Therefore, we will describe in more detail how to compose disjoint GIM classes. There are two different composition operations, the intensional and the extensional composition.

Only classes with equal extensions can be composed intensionally. If an inter-class integrity constraint is defined which describes a dependence between attributes of the involved classes then this integrity constraint becomes an intra-class constraint.

The extensional composition is more difficult than the intensional composition. Problems occur in cases when the classes to be composed are restricted by different integrity constraints. The question is how to compose the different integrity constraints according to the composition of the classes. In principle, there are two possibilities to compose different integrity constraints:

- *Conjunctive composition*
  The conjunctive composition of integrity constraints leads to integrity constraints which are stronger than the integrity constraints of an involved class. Therefore, not all objects of the involved GIM classes can be made available to global applications.

- *Disjunctive composition*
  Due the fact that disjunctively composed integrity constraints are weaker than their components, all object of the involved GIM classes can be made available to global applications. However, not each object, which can be inserted into the composed class, can be stored in each involved class on the integrated schema level.

If such conflicting integrity constraints occur the designer of the external schema has to decide whether this class composition is really necessary. If he insists on this composition then he has to choose one of the both variants to compose integrity constraints.

Up to now we have described the composition of a single class. Usually an external schema contains more than one class. If the data model of the external schema supports

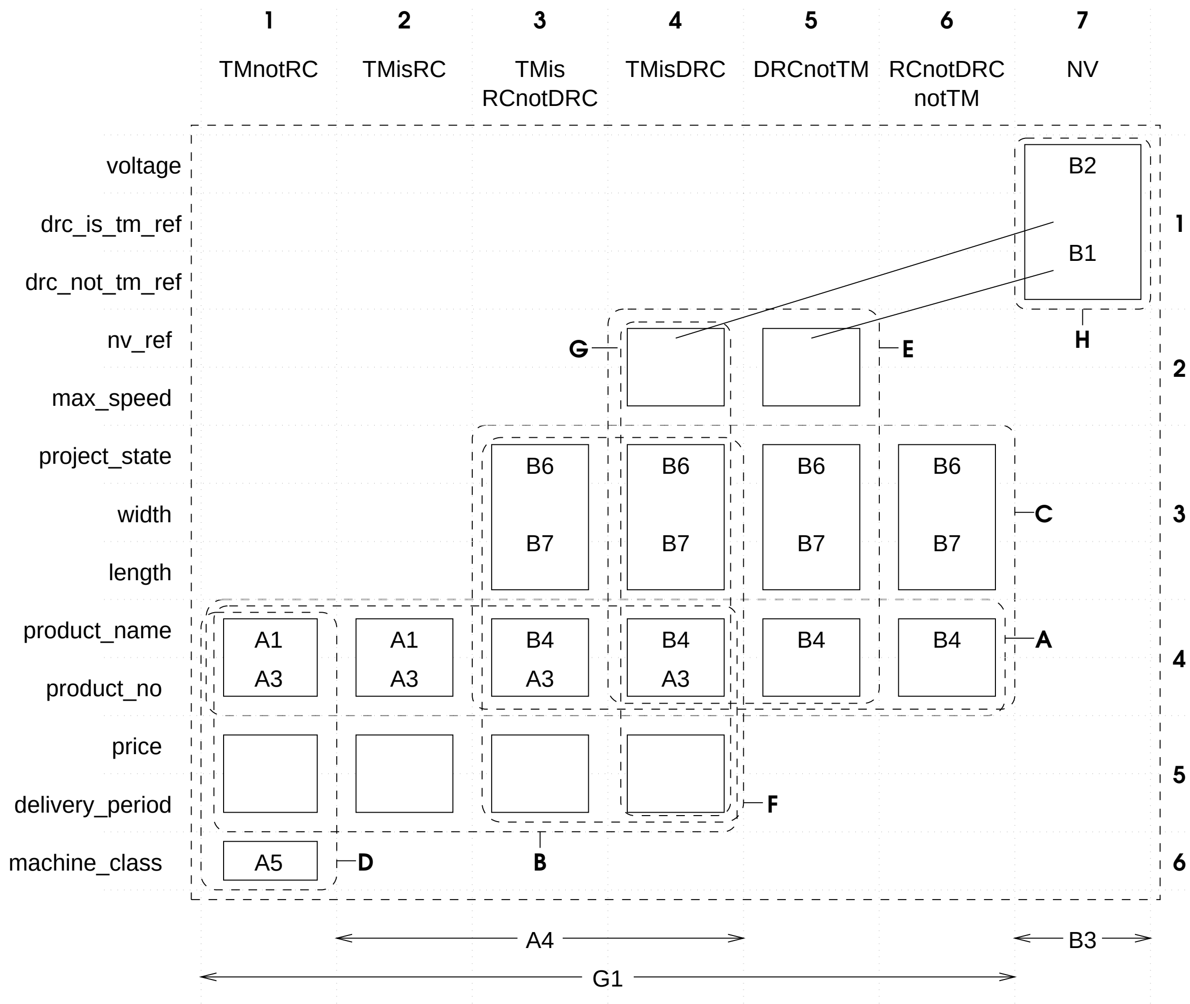

Figure 8.1: Derivation of a new external schema

the specialization concept then different classes can encompass intensionally and extensionally each other. *A superclass encompasses its subclasses extensionally and a subclass encompasses its superclasses intensionally.* Applying these rules to the integrated schema we are able to derive different inheritance hierarchies.

After having described the principles of composing GIM classes we introduce their interpretation in a GIM diagram as well as in table form. As mentioned earlier, in GIM diagrams each class is represented by a rectangle. Hence, composing classes means to enclose rectangles to a greater rectangle. Classes in a specialization relationship are represented by overlapping rectangles in correspondence to the rules defined before. Due to the graphical restrictions of GIM diagrams we propose an additional notation to express relations between GIM classes and composed classes. In a table form composed classes are related to their extensional and intensional coordinates. Each combination of extensional and intensional coordinates in a composed class represents a GIM class. The

Derived Classes		*Extensions*	*Intensions*	*Integrity Constraints*
A	**Abstract Class**	1,2,3,4,5,6	4	X1,X2,X3
B	**TM**	1,2,3,4	4,5	–
C	**RC**	3,4,5,6	3,4	X8,X9
D	**TMnotRC**	1	4,5,6	X5
E	**DRC**	4,5	2,3,4	–
F	**TMisRC**	3,4	3,4,5	–
G	**TMisDRC**	4	2,3,4,5	–
H	**NV**	7	1	X4,X6,X7

Figure 8.2: Derived classes and integrity constraints

set of intensional coordinates of a superclass has to be a subset of the intension set of its subclass. Vice versa, the inverse subset condition has to be valid to the corresponding extension sets. Besides the information about composing relationships the table form contains additional information about intra-class integrity constraints defined on the composed classes.

Now we want to apply our methodology to the running example. The derivation of an object-oriented external schema from the integrated schema which is enriched by the global integrity constraint $\mathrm{G1}_{GIC}$ will be shown. In order to derive the external schema we only use composing operations. The classes of the integrated schema are composed to external classes as it is shown in Figure 8.1 and in Figure 8.2.

In the GIM diagram each composed class is surrounded by a dashed rectangle and is marked with a character. The same character is used in the GIM table. Furthermore, the intensional and extensional coordinates in the GIM table correspond to the coordinates depicted in the diagram. Following the rules of deriving specialization relations between classes we are able to derive the inheritance hierarchy (cf. Figure 8.3) from the diagram and the table. As mentioned above, during the class composition conflicting integrity constraints can occur. This happens by the composition of all GIM classes with the intensional coordinate 4 to the abstract class A. The integrity constraints defined on the classes with the extensional coordinates 1 and 2 are different to the integrity constraints defined on the classes with the extensional coordinates 3 and 4 which in turn are different to the integrity constraints defined on the classes with the extensional coordinates 5 and 6. The different integrity constraint ($\mathrm{A1}_{IIC}$, $\mathrm{A3}_{IIC}$, $\mathrm{B4}_{IIC}$, $\mathrm{G1}_{GIC}$) are composed conjunctively. As a result, the following integrity constraints are defined on the external class A:

$\mathrm{X1}_{XIC}$[product_name][A]: length(product_name)$\leq$ 20 (IC_{11}, intra-class, static)

$\mathrm{X2}_{XIC}$[product_no][A]: product_no is not null (IC_{11}, intra-class, static)

$\mathrm{X3}_{XIC}$[product_no][A]: product_no is unique (IC_{m1}, intra-class, static)

Please note that in the external data model the explicit integrity constraint $\mathrm{X1}_{XIC}$ can be expressed as model-inherent data type `string<20>`.

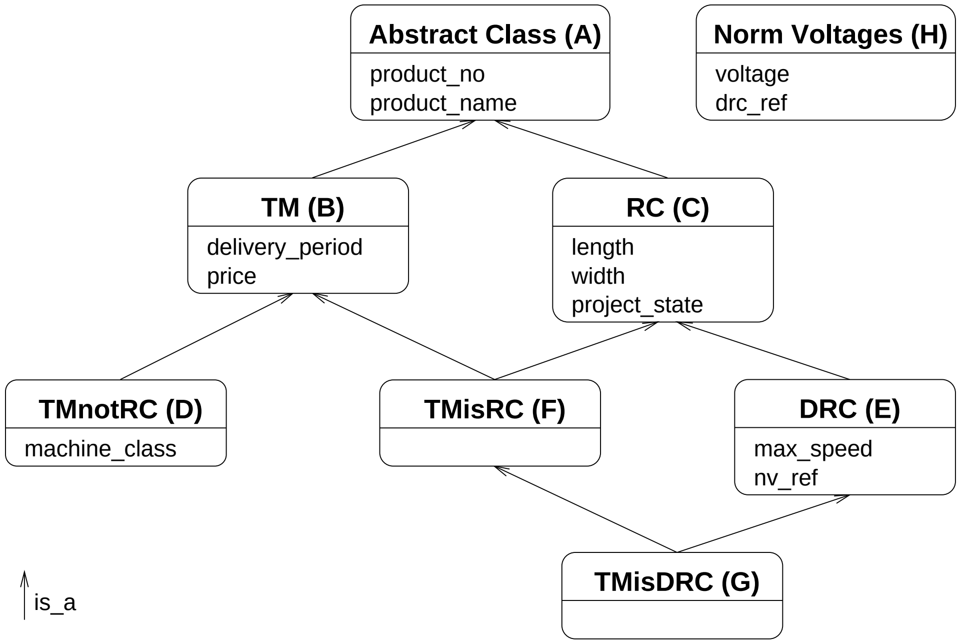

Figure 8.3: Derived object-oriented external schema

The composition to the class E necessitate a further transformation step. The reference attributes `nv_ref` of the GIM classes with the extensional coordinate 4 and 5 are merged to one reference attribute of the composed class. This merging leads to a merging of the inverse reference attributes `drc_is_tm_ref` and `drc_not_tm_ref` of the class `NV` to the reference attribute `drc_ref`. Finally, we have to transform the integrity constraint $B1_{IIC}$ as follows:

$X4_{XIC}$[drc_ref][NV]: card(drc_ref)>0 (IC_{11}, intra-class, static)

The remaining of the integrity constraints of integrated schema ($A5_{IIC}$, $B2_{IIC}$, $B3_{IIC}$, $B6_{IIC}$, $B7_{IIC}$) can be adopted directly:

$X5_{XIC}$[machine_class][TMnotRC]: machine_class≠'roller conveyor' (IC_{11}, intra-class, static)

$X6_{XIC}$[voltage][NV]: voltage is not null (IC_{11}, intra-class, static)

$X7_{XIC}$[voltage][NV]: voltage is unique (IC_{m1}, intra-class, static)

$X8_{XIC}$[project_state][RC]: project_state:in_design⇒in_construction⇒completed (IC_{11},intra-class,transitional)

$X9_{XIC}$[length,width][RC]: length>15*width (IC_{1n}, intra-class, static)

Chapter 9

Discussion and Outlook

By means of a running example we have shown our methodology for federating heterogeneous database schemata in a comprehensible way. Because our methodology even captures the treatment of integrity constraints during the federation process, the presentation has especially focused on that aspect. From our point of view, constraints are an important part of schemata and, therefore, they have to be considered during the whole process. Especially, the evolution of integrity constraints must be considered as intrinsic to federating heterogeneous databases. Based on our classification of integrity constraints, we have shown typical patterns for this evolution. Our approach is based on a *Generic Integration Model* (GIM) which offers a small number of orthogonal concepts. Thereby, GIM provides a high degree of flexibility in the federated database design process.

In the following we briefly discuss several aspects of our methodology and compare our approach with some related work. Finally, we conclude and give an outlook on ongoing work.

In Section 3 we presented a classification of integrity constraints for clarifying their evolution in the federation process. Considering integrity constraints during the transformation and integration process we can see that integrity constraints can change their characteristics. For several cases, this behaviour is obvious, for instance, certain kinds of constraints being inherent to object-oriented data models must be made explicit when transforming to GIM. However, not all model-inherent constraints of local database schemata must be made explicit, because there are also constraints being inherent to GIM (e.g., extensions in GIM are always non-overlapping). A typical behaviour of constraints is the change between inter-class and intra-class when we apply extensional and intensional decomposition or composition.

Besides the evolution of pre-existing integrity constraints, we have shown that we usually have to add new global constraints for resolving several kinds of conflicts arising in the federated database design process. However, it can easily be seen that there is not a unique and predictable choice of additional global constraints. The way of resolving the occurring conflicts also depends on the intended semantics of the federated schema and, thereby, on the intended global applications. Often, there is even not a consistent solution for all conflicts together. Therefore, we have to make compromises.

However, new integrity constraints can not only arise on the global level. Our methodology can also help in finding deficiencies w.r.t. the local database schemata. Often, not all possible (and reasonable) constraints are captured by the local schemata, possibly due to an incomplete database design for the component system. Deriving external schemata from the integrated schema in a way that they equal already existing external schemata on the local level, it can happen that we derive additional constraints for these new external schemata. In this way our approach can even help to complete the original local database design.

An important question arising during the transformation and integration process is which step in the process is the best step for adding new global constraints. From the case studies we made so far we conclude that it is the best approach to first build the integrated schema and then add new global constraints. During the integration a number of conflicts can arise which require additional global constraints (e.g., for controlling redundancy). These constraints must be introduced before deriving external schemata, because the external schemata may rely on these constraints.

Comparing our approach with other recent approaches it has to be noticed that we do not use an object-oriented data model as integration model. For instance, [GCS95] presents an object-oriented approach. For resolving conflicts, additional generalisation classes have to be introduced into the original class hierarchy. In [SPD92] an own object-oriented data model is introduced, and rules are given for the integration. Furthermore, several kinds of simple integrity constraints are considered. [KCGS95] discusses all kinds of conflicts in great detail without giving a complete methodology for transforming and integrating schemata.

Another approach coming closer to our ideas is described in [RPRG94, RPG95] where certain kinds of integrity constraints are considered. Our approach extends that one by allowing to deal with more general kinds of integrity constraints. Furthermore, our methodology is more flexible, for instance, by giving more concrete help in deciding how to resolve conflicts. This is due to the fact that we consider constraints as first class elements in the integration process. Thereby, we are able to fill some gaps left open by other proposals.

The integration of heterogeneous database schemata into one federated schema brings together a number of problems already known from other areas. Due to the distribution of data there are certain problems related to distributed database design (see e.g. [ÖV94]). Furthermore, the integration process is faced with problems of view integration (for an overview see e.g. [BLN86]). Heterogeneity and autonomy of the component systems even complicate this situation.

As already mentioned before, a number of methodologies have been proposed for getting a grasp of different aspects of the integration problem. Our methodology extends most other proposals by explicitly considering integrity constraints during the whole transformation and integration process. Respecting constraints from the very beginning is useful for resolving most of the conflicts arising during the federation process. Taking the constraints into account shows in several cases which is the best way to resolve a conflict or at least gives strong indications for reasonable solutions.

We are currently working on a formalisation of the transformation, integration and derivation steps based on GIM. Thereby, tool support for the integration of heterogeneous database schemata will be possible. For describing integrity constraints in this paper, we have used an ad-hoc notation capturing a wide spectrum of constraints. In a forthcoming paper, we will provide a formally defined language for specifying constraints and investigate the limitations w.r.t. the treatment of constraints in our approach. Together with the formalisation we are going to consider the resolution of integration conflicts in more detail. For several kinds of conflicts the possible solutions seem to be obvious, for others we have several ideas which have first to be verified by more case studies and second to be brought together with the formalisation.

Bibliography

[BFN94] R. Busse, P. Fankhauser, and E. J. Neuhold. Federated Schemata in ODMG. In J. Eder and L. A. Kalinichenko, editors, *Extending Information Systems Technology – Proc. of the 2nd Int. East/West Database Workshop, Klagenfurt, Austria*, pages 356–379. Workshops in Computing, Springer-Verlag, September 1994.

[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, 1991.

[BHP92] M. Bright, A. Hurson, and S. Pakzad. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, 25(3):50–56, March 1992.

[BKNW91] T. Barsalou, A. Keller, Siambela N., and G. Wiederhold. Updating Relational Databases through Object-Based Views. In J. Clifford and R. King, editors, *Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data, Denver, Colorado, SIGMOD RECORD 20(2)*, pages 248–257. ACM Press, June 1991.

[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.

[BS95] M. L. Brodie and M. Stonebraker. *Migrating Legacy Systems*. Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[Cat94] R. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann Publishers, San Mateo, CA, 1994.

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

[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.

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

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

[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*, pages 505–512. IEEE Computer Society Press, March 1995.

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

[JQ92] H. V. Jagadish and X. Qian. Integrity Maintenance in an OODB. In L.-Y. Yuan, editor, *Proc. of the 18th Int. Conf. on Very Large Data Bases (VLDB'92), Vancouver, Canada*, pages 469–480. Morgan Kaufmann Publishers, August 1992.

[KC86] S. N. Khoshafian and G. P. Copeland. Object Identity. In N. Meyrowitz, editor, *Proc. of the 1st Int. Conf. on Object Oriented Programming Systems, Languages and Applications (OOPSLA'86), Portland, Oregon, SIGPLAN Notices 21(11)*, pages 406–416. ACM Press, November 1986.

[KCGS95] W. Kim, I. Choi, S. Gala, and M. Scheevel. On Resolving Schematic Heterogeneity in Multidatabase Systems. In W. Kim, editor, *Modern Database Systems*, pages 318–337. ACM Press, 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. Language Features for Interoperability of Databases with Schematic Discrepancies. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan*, pages 144–151. IEEE Computer Society Press, April 1991.

[LGS94] U. W. Lipeck, M. Gertz, and G. Saake. Transitional Monitoring of Dynamic Integrity Contraints. *Bulletin of the IEEE Technical Committee on Data Engineering*, 17(2):38–42, June 1994.

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

[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*, pages 294–301. IEEE Computer Society Press, April 1993.

[ÖV94] M. T. Özsu and P. Valduriez. Distributed Data Management: Unsolved Problems and New Issues. In T. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, chapter 10, pages 512–544. IEEE Computer Society Press, 1994.

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

[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, April 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.

[RS94] E. Radeke and M. H. Scholl. Framework for Object Migration in Federated Database Systems. In *Proc. of the 3rd Int. Conf. on Parallel and Distributed Database Systems (PDIS'94), Austin, USA*. IEEE Computer Science Press, September 1994.

[SCG91] F. Saltor, M. Castellanos, and M. Garcia-Solaco. Suitability of Data Models as Canonical Models for Federated Databases. In J. Clifford and R. King, editors, *Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data, Denver, Colorado, SIGMOD RECORD 20(2)*, pages 44–48. ACM Press, June 1991.

[Sch95] I. Schmitt. Flexible Integration and Derivation of Heterogeneous Schemata in Federated Database Systems. Preprint Nr. 10, Fakultät für Informatik, Universität Magdeburg, November 1995.

[SJH93] G. Saake, R. Jungclaus, and T. Hartmann. Application Modelling in Heterogeneous Environments Using an Object Specification Language. In M. Huhns, M. P. Papazoglou, and G. Schlageter, editors, *Int. Conf. on Intelligent & Cooperative Information Systems (ICICIS'93)*, pages 309–318. IEEE Computer Society Press, 1993.

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

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

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

[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), Vienna, Austria*, pages 4–18, May 1995.