Schema Integration with Integrity Constraints

Stefan Conrad Michael Höding Gunter Saake Ingo Schmitt Can Türker

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

Abstract. In this paper we discuss the use and treatment of integrity constraints in the federated database design process. We consider different situations occurring frequently in the schema transformation and schema integration process. Based on that, general rules are given which describe the correct treatment of integrity constraints. Other proposed integration approaches do not consider integrity constraints at all or are restricted to special kinds of constraints. Therefore, our approach can be used to extend or complete existing integration methodologies.

1 Introduction

Considering the current situation in enterprises we can discover that many different information systems are used in different parts of the enterprise. Furthermore, there is an increasing need to get a uniform access to the heterogeneous databases lying behind those information systems. The aim is to obtain a federated database system in the sense of [SL90]. Besides a uniform access to data stored in different database systems, migration of applications and data from one database system to another one is an additional motivation for building federated database systems [RS94].

The integration of heterogeneous databases is a well-known problem. A large number of approaches for integrating the local schemata of such databases have been proposed up to now, e.g. [SPD92, BFN94, RPRG94, GCS95]. Unfortunately, most approaches do not take integrity constraints into consideration, or if they respect constraints they only capture certain kinds of constraints [RPG95].

The aim of this paper is to give a survey about the problems arising when integrating schemata and their integrity constraints. Due to the heterogeneity of data models used for local schemata, the first step of schema integration is usually a transformation of the local schemata into schemata expressed in a common data model. Together with the schemata the integrity constraints must be transformed. Obviously, there are often a lot of constraints inherent to the data model used by a local schema. These constraints must be made explicit by the transformation, if such kinds of constraints are not inherent to the common data model.

After having removed the heterogeneity w.r.t. data models, the integration can take place. All proposed methodologies capture the integrity constraints inherent to the common data model used for integration. This is due to the fact that these constraints are already captured by the semantics of the modeling concepts of the common data model. The integration rules given by integration methodologies respect this semantics. If we now consider the constraints explicitly given for the schemata to be integrated, we can see that there are many cases where we can simply transfer the local constraint to the integrated schema. However, we often have to face situations where this is not possible.

In this paper we discuss such cases and point out possible and reasonable solutions. For presentation we use the *Generic Integration Model* GIM [SS96, SC97]. GIM provides an intermediate graphical notation in which the schemata to be integrated can be represented in a normalized way. We extend this notation in a straightforward way for representing integrity constraints.

The remainder of this paper is organized as follows: First, we briefly introduce the concepts of GIM and extend its graphical notation for capturing integrity constraints. In Section 3 we introduce an example which we use in the subsequent sections for discussing the transformation and integration of constraints. Details on schema transformation and integration using GIM can be found in Section 4 and 5. There we especially investigate solutions for integrating conflicting constraints. Finally, we conclude with discussing some important aspects of our approach and with a comparison with related work.

2 Foundations

In this section, we briefly recall the basic steps of the design process of a federated database and introduce the Generic Integration Model GIM which we use as an intermediate representation layer from which object-oriented integrated schemata may be easily derived.

According to [SL90], the existing local schemata of the different (heterogeneous) component databases 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 component databases, are transformed into a common data model. After this transformation step the data model heterogeneity is overcome. The following schema integration step itself is very difficult and is further 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 removed. In the last design step, external schemata which are specific views on a federated schema are derived.

For best support of the integration process we have designed a data model called the Generic Integration Model. In GIM, only schema information which is necessary for the integration process can be expressed. The following list summarizes the basic concepts of GIM:

- *Simple data types*
- *Non-overlapping class extensions*
- *Object identifier*
- *Bidirectional binary references*
- *Integrity constraints similar to SQL2*

GIM is not intended to be used as an interface to global applications. Rather, GIM is an intermediate representation form which is only used for integration purposes. A detailed description of GIM can be found, for instance, in [SS96, SC97]. Here, we introduce only the notation of GIM diagrams which are used to model GIM schemata (cf. Figure 1).

The horizontal dimension of a GIM diagram refers to the extensional aspect of a class 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. The names

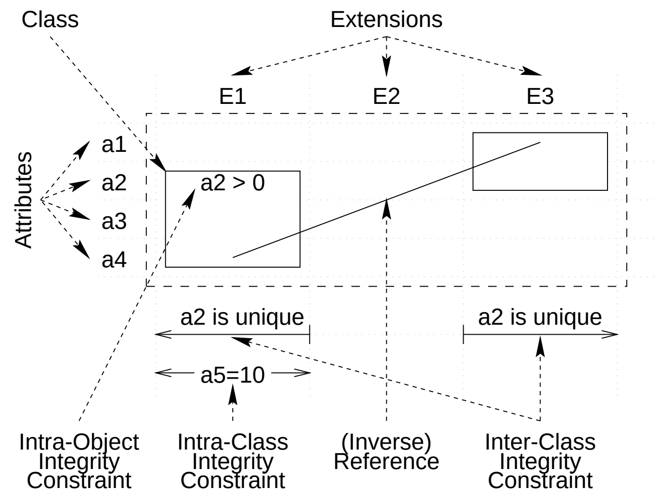

Figure 1: Diagram conventions

of the extensions and the names of the attributes are located on the top and on the left of the diagram, respectively.

In GIM, we distinguish three categories of integrity constraints. The first one, called *intra-object* integrity constraints refers to constraints which may be validated on exactly one object, e.g. the `not null` condition. This kind of constraints are represented within the respective GIM class and define possible values for a set of attributes of an object. The second category of constraints concern the *intra-class* integrity constraints. These constraints may be only validated by considering the whole extension of a class. The uniqueness condition, for instance, falls in this category. Finally, there are *inter-class* integrity constraints which refer to objects of several classes. Both, intra-class and inter-class integrity constraints are located below the GIM diagram as depicted in Figure 1.

3 A Running Example

In this section, we introduce an example which we use in the following sections to illustrate our approach for the design of federated databases.

We assume that there is a company with two departments, the sales and the construction department. Both departments have their own informations systems which are based on two different database systems. The sales department stores information about the commercial aspects of the products in the sales database. As depicted in Figure 2, this database contains a local schema which consists of a class transport machines (`TM`). The construction department of the company is responsible for the construction and production of roller conveyors (`RC`) which are a special kind of transport machines. In consequence, the information system of this department, which relies on the construction database, deals with data concerning the properties of roller conveyors. The local schema of this database consists of two classes `RC` and `DRC`, whereas the latter one represents driven roller conveyors which are special roller conveyors.

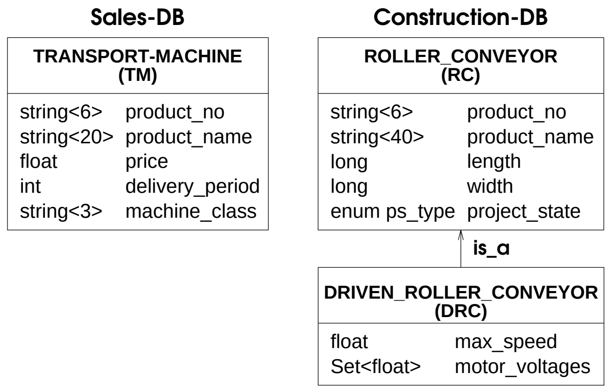

Figure 2: Existing local schemata

Moreover, suppose that there are several integrity constraints defined on these local schemata. Table 1 summarizes all existing constraints.

Attributes	Ext.	Description	Remark
Sales-DB			
product_no	TM	key and not null	intra-class
Construction-DB			
product_no	RC	key	intra-class
length, width	RC	length<15∗width	intra-object
project_state	RC	in_design⇒in_construction⇒completed	intra-object

Table 1: Existing local integrity constraints

Finally, let us have a look at the extensions of the local classes. Due to the fact that there are transport machines in the sales database which are also roller conveyors in the construction database, we have to deal with overlapping extensions of classes which are stored 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 is depicted in Figure 3 and will be further considered during the federated database design process described in the following sections.

4 Schema Transformation into GIM

In this section, we briefly present the transformation of the two schemata of our example into the Generic Integration Model. Thereby, we especially focus on the evolution of the integrity constraints during the transformation process.

[1] We assume that objects stored in both databases can be identified by equal product numbers.

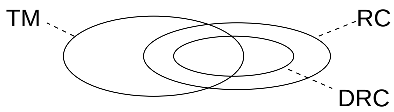

Figure 3: Overlapping extensions of example classes

4.1 Transformation to Simple Data Types

In our example, the first normal form is violated by the definition of the set-valued attribute `motor_voltages` in class `driven_roller_conveyor`. Therefore, similar to the normalization process in the relational database design process set-valued attributes have to be transformed into "independent" classes. The result of this normalization step is a new class `NV` (norm-voltages) which contains the reference attribute `drc_ref` and the integrity constraint `B1` which defines the existential dependency of class `NV` from class `DRC`. In order to identify different norm voltages, we have to introduce the constraints `B2` and `B3` which ensure the primary key condition (cf. Table 2).

Please note that some problems may occur if there is no isomorphic mapping between local data types and GIM data types. In this case, additional integrity constraints must be derived to restrict the domain of a GIM data type, e.g. upper bounds, lower bounds, accuracy bounds or enumerated values. In our example, we have generated the new integrity constraints `A1` and `B4` in order to restrict the length of the character strings which represent the product names (cf. Table 2).

4.2 Non-overlapping Class Extensions

The requirement that all class extensions should be non-overlapping 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 a violation can be resolved by extensional decomposition. If the extensions of the classes `A` and `B` overlap, then we generate three classes `A`$-$`B`, `B`$-$`A` and `A`$\cap$`B` (using the usual set operations $-$ and $\cap$) which have non-overlapping extensions. 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`$\cap$`B` contains all attributes of both classes `A` and `B`.

Moreover, after the extensional decomposition has been carried out, the relevant integrity constraints have to be transformed. Intra-object integrity constraints can be adopted directly to the classes which are the result of the extensional decomposition. In contrast, intra-class integrity constraints have to be transformed into inter-class integrity constraints covering all resulting classes. Please note that integrity constraints originally defined on a single class change to a inter-class constraint after the extensional decomposition because the original class is subdivided in several classes.

In our example, the result of the decomposition of the classes `roller_conveyor` and `driven_roller_conveyor` are the classes `DRC` and `RCnotDRC`. Please notice that the extension of class `RCnotDRC` contains all roller conveyors which are *not* driven roller conveyors. In contrast, the extension of class `DRC` contains only the driven roller conveyors. 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. Furthermore, the key constraint of the class roller_conveyor is transformed into the inter-class unique constraint B5 and has to be checked against the union of the extensions of classes RCnotDRC and DRC. The transformed schemata are depicted in Figure 4 as GIM diagrams. The integrity constraints for these component schemata are listed in Table 2.

No	Attributes	Extensions	Description	Remark
A1	prod_name	TM	length<20	intra-object
A2	prod_no	TM	unique	intra-class
A3	prod_no	TM	not null	intra-object
B1	drc_ref	NV	card(drc_ref>0)	intra-object
B2	voltage	NV	not null	intra-object
B3	voltage	NV	unique	intra-class
B4	prod_name	RCnotDRC, DRC	length<40	intra-object
B5	prod_no	RCnotDRC ∪ DRC	unique	inter-class
B6	project_state	RCnotDRC, DRC	in_design⇒in_construction⇒completed	intra-object
B7	length, width	RCnotDRC, DRC	length<15∗width	intra-object

Table 2: Integrity constraints of the component schemata

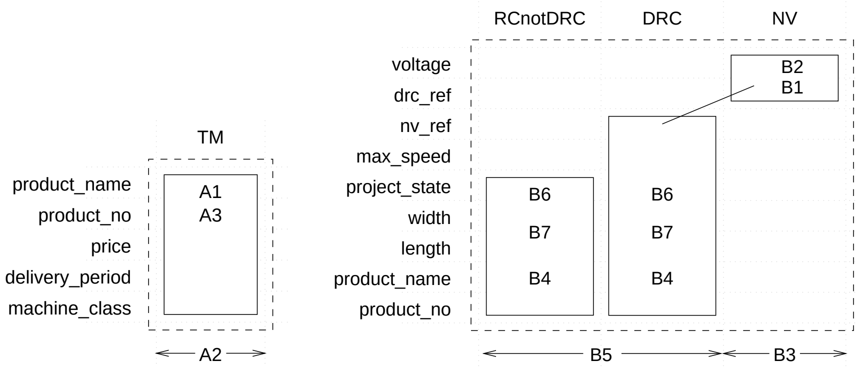

Figure 4: Schemata transformed to GIM

In conclusion, we briefly summarize the basic principles of the evolution of integrity constraints during the transformation process. In the following, we list two important rules which may be helpful for a tool-assisted design process:

Rule 1: Intra-object integrity constraints have to be adopted to the classes resulting from the extensional decomposition process.

Rule 2: Intra-class integrity constraints are always transformed into inter-class integrity constraints of the extensional decomposed classes.

Each transformation of integrity constraints must obey these rules.

5 Schema Integration

After all local schemata have been transformed into the common data model the data model heterogeneity is resolved. In order to overcome the schema heterogeneity the component schemata have to be integrated. Schema integration is a very complex task. Many kinds of conflicts between two schemata to be integrated can occur which have to be resolved. A prerequisite for conflict resolution is that all conflicts can be detected. Such a detection cannot be done automatically, rather it must be assisted by an integration tool. We propose the following steps for the integration process:

1. Resolving meta conflicts
2. Resolving attribute conflicts
3. Resolving intensional conflicts
4. Resolving extensional conflicts
5. Derivation of an object-oriented integrated schema

After having carried out step one to four all conflicts are resolved in the GIM presentation. As result from these four steps we obtain the *normalized* schema which is used to derive an object-oriented integrated schema.

In order to improve the comprehensibility of the complex integration steps we show the results of the homogenizing operations by means of our example. We focus on the aspect how integrity constraints have to be dealt with during the integration steps.

5.1 Resolving Meta Conflicts

A meta conflict exists if objects of a class differ in values of a specific attribute and their corresponding objects of another schema differ in their class assignment. The values of the first schema correspond to classes of the other schema [KLK91].

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, only transport machines of which the attribute `machine_class` is set to "roller conveyor" may correspond to objects of the classes `RCnotDRC` and `DRC`.

Meta conflicts can be resolved by the transformation of the class being in the attribute variant to the corresponding class variants. This transformation is accompanied by splitting the class according to a comparison operation.

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 from "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`.

If a class is split by a comparison operation then new integrity constraints restricting attribute values are created. In our example, the value of the attribute `TMisRC.machine_class` is restricted to the fixed value "roller conveyor" (`A4`). This attribute can be dropped from the intension. The integrity constraint `A5` of the class `TMnotRC` prohibits the value "roller conveyor".

Furthermore, following Rule 1, the integrity constraints `A1` and `A3` are adopted to the classes. However, following Rule 2, the uniqueness constraint has to be transformed into the inter-class integrity constraint `A2`. `A2` is defined on the union of the extensions of the new classes. The resulting GIM classes of the construction database are depicted in Figure 5.

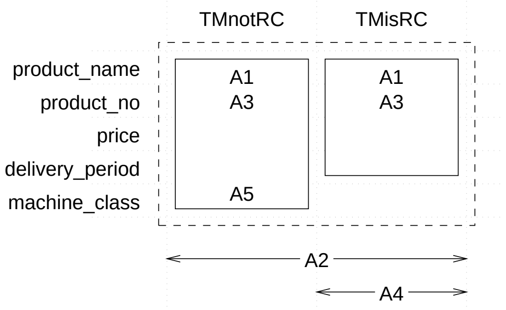

Figure 5: Sales schema after resolving the meta conflict

5.2 Resolving Attribute Conflicts

In order to arrange classes to each other in the intensional dimension, their attributes have to be compared to each other. Two or more attributes are considered as one attribute in the normalized 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 sometimes the attribute conflict can be transformed to an integrity constraint conflict by introducing new integrity constraints.

In our example we assume that all attributes, which have same names are semantically equivalent and compatible. An exception concerns the attribute `product_name`. Since the maximal string length of this attribute is restricted to 20 in the schema of the sales database and to 40 in schema of the construction database they are not compatible and no isomorphic mapping between them can be found. This attribute conflict was transformed to an integrity constraint conflict in Section 4 by introducing the integrity constraints `A1` and `B4`. We will show in Section 5.5 how to deal with conflicting integrity constraints.

Putting the transformed Classes into one Diagram

After having resolved the attribute conflicts, the classes of both schemata can be arranged vertically (intensionally) in one GIM diagram. From the extensional relationships between the local classes (cf. Figure 3) and the restruction operations up to now, the extensional relationships of the restructured classes can be derived automatically. The result is depicted in Figure 6.

We have now reached the point where the classes of both schemata can be put into one diagram (cf. Figure 7). In order to illustrate all extensional overlappings the extension of class `RCnotDRC` is located to the left and to the right of the extension of class `DRC` in Figure 7. This graphical decomposition must not be considered as a restructuring of the schema. It is necessary only for illustration purposes.

As we can see in Figure 7, the intensions and extensions of the classes overlap, e.g. classes `TMisRC` and `DRC`. However, an extensional overlapping is not allowed

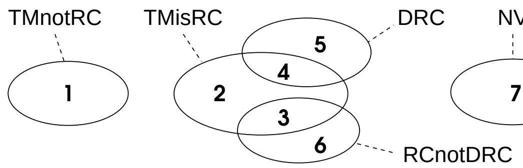

Figure 6: Extensions of the GIM classes

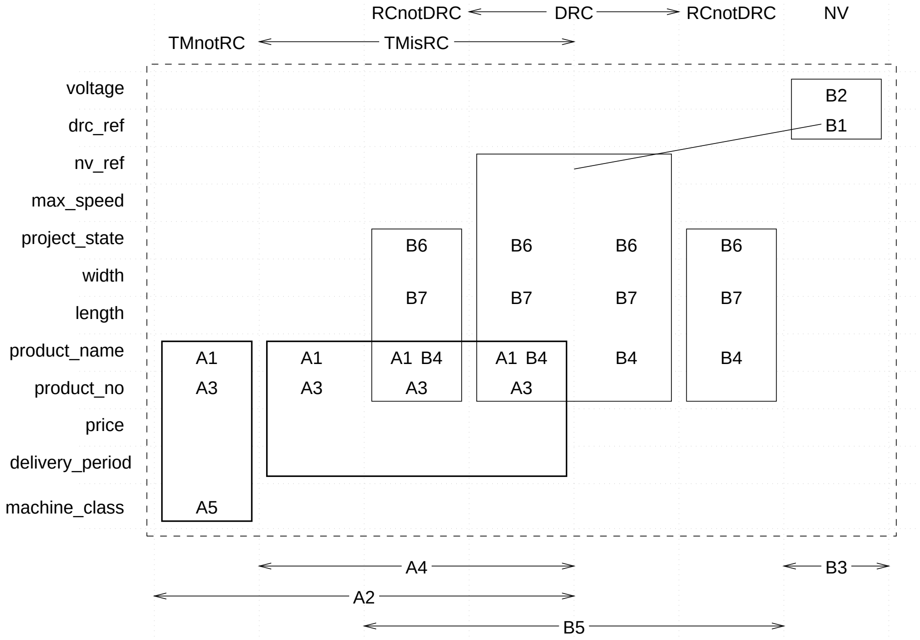

Figure 7: GIM classes in one diagram

in GIM. In the next subsections we show how the intensional and extensional overlappings have to be resolved.

5.3 Resolving Intensional Conflicts

We solve intensional conflict by intensional decomposition. Intensional decomposition means to group attributes in such a way that all attributes of a group are attributes of the same set of classes. Intensional decompositions are represented graphically by horizontal lines (cf. Figure 8). 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 change integrity constraints in our example.

5.4 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. This process is described in more detail in [SS96].

The result of this decomposition is illustrated in Figure 8. For example, since the classes `TMisRC` and `DRC` (cf. Figure 7) overlap, they are decomposed into the three classes `TMisRC, DRCisTM`, and `DRCnotTM`. The resulting class extensions have to be given names. In general, this is a quite difficult task. Therefore, we construct the names by means of set operations ('is' refers to intersection and 'not' refers to difference).

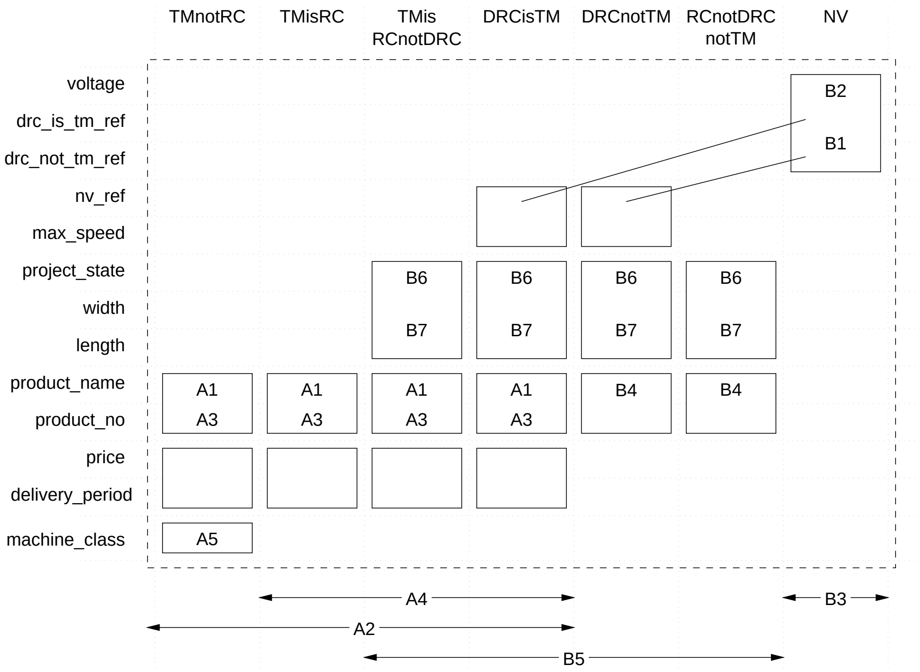

Figure 8: GIM schema after intensional and extensional decomposition

The decomposition of GIM classes causes changes of integrity constraints by applying rule one and two. For example the intra-object constraint B7 of the class `DRC` is adopted to the classes with the extensions `DRCisTM` and `DRCnotTM`.

However, Rule 1 and 2 do not cover all integrity constraint transformations. Each object of the class `DRCisTM` is an object of the sales database as well as of the construction database. Therefore, for the class `DRCisTM` the integrity constraints of the classes `DRC` and `TMisRC` have to be combined as a logical conjunction.

Rule 3: If the extension of a normalized GIM class is equivalent to the intersection of local class extensions, then the local intra-object and local intra-class integrity constraints must be combined as a logical conjunction.

Applying Rule 3 to the class `DRCisTM` the intra-object integrity constraints `A1`, `A3`, `B4`, `B6`, and `B7` are combined as a logical conjunction. Due to the fact that `A1` is more restrictive than `B4`, the constraint `B4` can be omitted.

The class `DRCisTM` hides redundancy. The values of the attributes `product_no` and `product_name` of this class are stored in both databases. In general, it can happens that the attribute values of the objects are different in both databases. The problem is how to deal with such inconsistencies. We propose to generate new integrity constraints defining equivalence conditions on all attribute values of such kind of objects. Such integrity constraints have to be hidden from global applications. In order to guarantee consistency with regard to these integrity constraints, 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 inconsistent data has to be made consistent. In other words the data heterogeneity has to be overcome.

Rule 4: If a global GIM class presents the intensional and extensional intersection of local classes from different databases, then redundancy exists. Such redundancy must be controlled by new integrity constraints which have to be hidden from global applications.

5.5 Derivation of Object-Oriented Integrated Schemata

Now we are able to generate an object-oriented integrated schema. A generation algorithm is described in [SS96]. Here, we only sketch the generation.

The classes of the object-oriented schema are generated by composing GIM classes to larger ones. 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 dependency 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 when the classes to be composed are restricted by different integrity constraints. We propose to combine such integrity constraints as a logical disjunction. Such composed integrity constraints are weaker than their components. Therefore, all objects of the involved GIM classes can be made available to global applications.

Rule 5: If a class of the object-oriented integrated schema is generated by an extensional composition of different GIM classes, then their integrity constraints must be combined as a logical disjunction.

Larger classes can be constructed not only by extensional compositions but also by intensional compositions.

Since the used object-oriented data model supports the specialization concept, 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 (cf. [SS96]).

In Figure 9 we show, how the GIM classes can be composed to classes of the object-oriented integrated schema. In the GIM diagram each composed class is surrounded by a dashed rectangle and is marked with a character. Following the rules of deriving specialization relations between classes we are able to derive the inheritance hierarchy (cf. Figure 10) from the diagram.

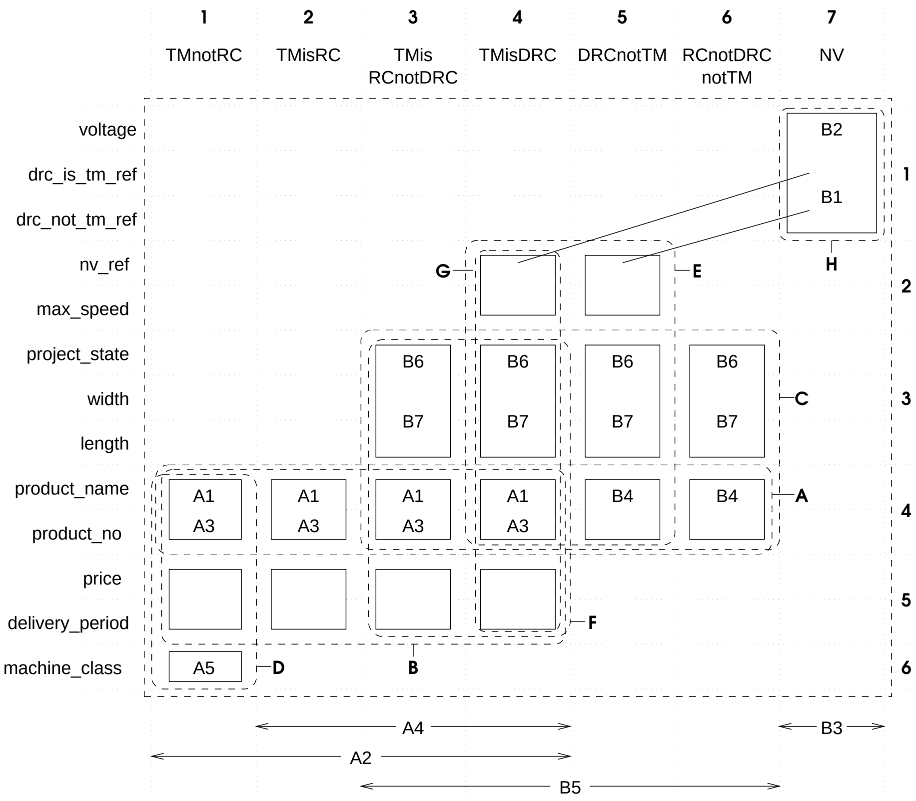

Figure 9: Derivation of an object-oriented integrated schema

Derived Classes		*Extensions*	*Intensions*	*Integrity Constraints*
A	**Product**	1,2,3,4,5,6	4	X1
B	**TM**	1,2,3,4	4,5	X2,X3,X4
C	**RC**	3,4,5,6	3,4	X1,X5,X6,X7
D	**TMnotRC**	1	4,5,6	X2,X3,X4,X8
E	**DRC**	4,5	2,3,4	X1,X5,X6
F	**TMisRC**	3,4	3,4,5	X2,X3,X5,X6
G	**TMisDRC**	4	2,3,4,5	X2,X3,X5,X6
H	**NV**	7	1	X9,X10,X11

Table 3: Derived classes and integrity constraints

Table 3 shows the name, the intension and the extension of each derived class in correspondence to Figure 9.

Table 4 contains the meaning of the new integrity constraints.

The composition to the class G necessitates 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`

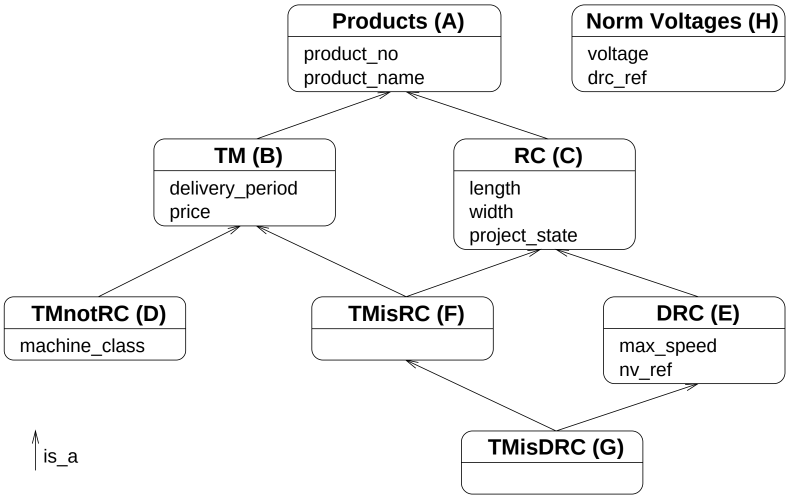

Figure 10: Derived object-oriented integrated schema

No	Attributes	Extensions	Description	Origin	Remark
X1	product_name	Product	length()<40	A1∨B4	intra-object
X2	product_name	TM	length()<20	A1	intra-object
X3	product_no	TM	not null	A3	intra-object
X4	product_no	TM	unique	A2	intra-class
X5	project_state	RC	in_design⇒in_construction⇒completed	B6	intra-object
X6	length, width	RC	length<15∗width	B7	intra-object
X7	product_no	RC	unique	B5	intra-class
X8	machine_class	TMnotRC	≠'roller conveyor'	A5	intra-object
X9	drc_ref	NV	card()>0	B1	intra-object
X10	voltage	NV	not null	B2	intra-object
X11	voltage	NV	unique	B3	intra-class

Table 4: Integrity constraints of the integrated schema

and `drc_not_tm_ref` of the class `NV` to the reference attribute `drc_ref`.

6 Discussion and Outlook

By means of a running example we have demonstrated basic principles of integrating schemata together with their integrity constraints. Although we have used our own integration model GIM for this purpose, the results can also be transferred to other schema integration approaches. In this way current schema integration approaches can be extended or completed by a comprehensive treatment of integrity constraints during the integration process. Constraints are an important part of database schemata and, therefore, they must be considered with equal rights during the whole integration process. In the following we briefly discuss several aspects of our approach and compare our work with related work. Finally, we give an outlook on ongoing and future work.

Considering integrity constraints during the transformation and integration process we can see that integrity constraints can change their characteristics. For several cases, this behavior is obvious, for instance, certain kinds of constraints being inherent to object-oriented data models must be made explicit when transforming to GIM (or a common data model in other approaches). 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 behavior 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.

We have found general rules for treating integrity constraints in the schema transformation and schema integration process. These rules mainly refer to decomposition and composition of classes and their corresponding extensions. We have presented the application of the rules by means of an example. It can be proved that the rules are correct. The proof can be based on a formal framework underlying the *Generic Integration Model* GIM. The correctness of our rules is, however, not only restricted to GIM. It is possible to apply these rules in the context of other integration approaches in which classes are decomposed and/or composed during the transformation and integration process.

In comparison with other recent approaches in the area of database integration it can be seen that our approach provides a useful extension which could be adopted by others. For instance, [GCS95] presents an object-oriented integration approach. For resolving conflicts, additional generalization classes have to be introduced into the original class hierarchy. Explicit constraints are not used. In [SPD92] an own object-oriented data model is introduced, and rules are given for the integration. Furthermore, some simple integrity constraints are considered. [KCGS95] discusses a large number of conflicts between schemata in great detail, however, the treatment of explicit constraints is not investigated. [VA96] proposes to consider data integration before schema integration. For data integration several kinds of integrity constraints are considered (for a classification of integrity constraints see e.g. [CQ92]). The integration of constraints is discussed with regard to conflicts for data integration. 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. We are considering constraints as first class elements in the integration process. Thereby, we are able to fill some gaps left open by other proposals.

We are currently working on a formalization for the integration of integrity constraints. This formalization is to capture all cases in which the integrated constraints can be derived automatically. In addition, we intend to provide tool support for schema integration with constraints. Several parts of a tool supporting schema integration using GIM have already be implemented. Furthermore, we have developed a prototype of a federated database system for investigating the influence of different kinds of conflict resolution on the dynamic behavior of a federated system. Especially, with regard to integrity constraints and different ways of integrating them we intend to develop criteria which will help in choosing the best way of integrating schemata together with their constraints.

References

[BFN94] R. Busse, P. Fankhauser, and E. J. Neuhold. Federated Schemata in ODMG. In J. Eder and L. A. Kalinichenko, eds., *East/West Database Workshop, Proc. 2nd Int. Workshop*, pages 356–379, Springer, 1994.

[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.

[CQ92] R. Cooper and Z. Qin. A Graphical Data Modelling Program with Constraint Specification and Management. In P. M. D. Gray and R. J. Lucas, eds., *Advanced Database Systems, Proc. 10th British National Conf. on Databases*, pages 192–208, LNCS 618, Springer, 1992.

[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.

[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, eds., *Proc. 3rd Int. Conf. on Cooperative Information Systems (CoopIS'95)*, pages 19–31, May 1995.

[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 521–550, ACM Press, New York, 1995.

[KLK91] R. Krishmamurthy, W. Litwin, and W. Kent. Interoperability of Heterogeneous Databases with Schematic Discrepancies. In Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, eds., *Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems*, pages 144–151, IEEE CS Press, 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 & 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. 3rd Int. Conf. on Parallel and Distributed Database Systems (PDIS'94)*, IEEE Computer Science Press, 1994.

[SC97] I. Schmitt and S. Conrad. Restructuring Class Hierarchies for Schema Integration. In *Proc. 5th Int. Conf. on Database Systems for Advanced Applications (DASFAA'97)*, World Scientific Press, Singapore, 1997.

[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.

[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.

[SS96] 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. 15th Int. Conf.*, pages 195–210, LNCS 1157, Springer, 1996.

[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, eds., *Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 425–435, Morgan Kaufmann, 1996.