Operations of Homogenizing and Heterogenizing in Federated Database Systems [1]

Ingo Schmitt

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

Email: schmitt@iti.cs.uni-magdeburg.de

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

September 1996

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

Abstract

The federation of heterogeneous database systems is still a challenge for the database system research. A main topic in federated database systems is the process of overcoming the heterogeneity of component schemata and the process of deriving heterogeneous external schemata for global applications. Semantically rich data models increase the complexity of managing heterogeneity. In this paper heterogeneity of component schemata is overcome by homogenizing operations in a semantically poor data model. Heterogeneous external schemata can be derived from the federated schema by heterogenizing operations which are inverse to the homogenizing operations. Due to the homogenizing by decomposing of classes and heterogenizing by composing of classes, flexible schema transformations and integration is possible.

Keywords: federated database design, generic integration model, homogenizing, heterogenizing

Contents

Chapter 1

Introduction

In many large organizations, different legacy data management systems are maintained and operated. These data management systems and the data managed by them have developed independently from each other. The data management systems are often database management systems or merely file-based systems differing in several aspects such as data model, query language, system architecture, etc. as well as the structure and semantics of the data managed. In this context the term 'heterogeneity of the data management systems' is commonly used. As a rule, applications for specific needs continue to be based on such systems. More recent applications often require access to the distributed databases but their implementation fails due to heterogeneity. *Federated database management systems* (FDBMS) are designed to overcome this problem. The need for FDBMS and its architecture is explained in greater detail in [SL90]. FDBS's integrate data of legacy DBMS's only virtually, that is 'only' the local schemata are integrated. In an FBDS different kinds of schemata have to be maintained and structured thus forming the schema architecture proposed in [SL90]:

1. *Local schemata* are the schemata provided by the existing local databases and are based on specific data models.

2. *Component schemata* are the local schemata which are transformed into the data model of the federated schema. On this level the data model heterogeneity is overcome.

3. The distinction between the data allowed to be federated and that which is not allowed to be federated, is defined by *export schemata*. Only exported schema elements and their data are accessible by the federation layer.

4. The *federated schema* represents the central, homogeneous schema and is generated by an integration process of the export schemata.

5. The *external schemata* serve as specific interfaces for global applications. A specific interface is a schema which represents either a specific view or a federated schema which is based on a specific data model or a combination of both.

Mapping information needed for schema transformations, integration and auxiliary information has to be stored in a global DBMS.

An FDBS has to deal with the existing heterogeneity of the component DBS's and has to support heterogeneity on the global level. Heterogeneity on the global level is caused by different views of global applications. Local heterogeneity is overcome by the integration process resulting in the federated schema. Global heterogeneity is achieved by mapping from the federated schema. We shall henceforth refer to the transformation to heterogeneous, external schemata as *heterogenizing*. In order to move a local application to global level, the FDBS has to enable the creation of an external schema which is equivalent to the local schema.

[Sch95] proposes the use of a semantically poor data model in the form of a canonical data model instead of a semantically rich data model, e.g. an object-oriented data model such as in [PBE95, Här94, RBB+95, TS93, BFHK94, CL94, GCS95, KKM93]. This suggestion is adopted here. The homogenizing and heterogenizing operations are based on the semantically poor data model GIM introduced in [Sch95] and formally defined in [SS96b]. In [SS96a, SS96c] a more detailed description of integrating different inheritance trees using GIM is given. The managing of integrity constraints in an FDBS is explained in [CHJ+96].

This work introduces and explains different operations to homogenize heterogeneous schemata into a federated schema and then in turn to heterogenize it into external schemata.

The next chapter gives some fundamental principles of the proposed approach. In Chapter 3 the homogenizing operations and in Chapter 4 the heterogenizing operations will be described. Finally, Chapter 5 concludes by summing up and giving a short outlook to the possible future developments in the area.

Chapter 2

Heterogeneity in FDBS

The heterogeneity can be classified into three different levels:

- data model;
- schema;
- data.

This work focuses on schema heterogeneity. Unfortunately, the heterogeneity of the schema level is dependent on the heterogeneity of the data model level. In [Kim95, SP91, Dup94] different classifications of schema conflicts dependent on different data models are proposed. Semantically rich data models cause more heterogeneity on the schema level than semantically poor data models. This heterogeneity on schema level in turn increases the complexity of the integration process. On the other hand, an FDBS has to support such semantically rich data models. The author believes that a semantically rich data model is only suitable to represent the specific view of an application instead of representing the conceptual schema of a DBMS in accordance with the three-level architecture [TK78]. Thus, the view proposed here is that semantically rich data models should only be supported in local and external schemata. Here the semantically poor data model GIM functions as the canonical data model. The following list shows the basic concepts of the GIM:

- *Simple datatypes*: The concept of a relation in the first normal form is adopted from relational database management systems in order to define the intension of a class. A definition of derived (calculated) attributes is not supported;
- *Non-overlapping class extensions* means that the extensions of any two classes are either disjunctive or identical. Subset relations or partial overlapping between class extensions are not allowed. Therefore, integration by upward inheriting is not applicable;
- *Object identifier* (OID) known from object-oriented systems [KC86];

- *Bidirectional references* (binary relationships) using the OID concept: A reference expresses the relationship between exactly two objects. The GIM supports different cardinalities (1:1, 1:n, m:n) of reference types between classes (relations). All references in the GIM are bidirectional. Each reference has an inverse reference in the GIM. References are considered as attributes of a class. Only such attributes can be multi-valued.
- *Integrity constraints*: GIM supports integrity constraints similar to SQL2. An essential type of an interclass integrity constraint is the existential dependency between objects.

The following steps are necessary to establish an FDBS using the GIM. We do not take into account export schemata:

1. *Transformation*: The local schemata are transformed into component schemata which are based on the GIM. Now the heterogeneity on the data model level is overcome. Furthermore, due to the semantical poverty of the GIM view-dependent information expressed in semantically richer data models is lost which decreases heterogeneity on schema level. However, this information is captured in suitable data structures.
2. *Integration*: The remaining heterogeneity on the schema level is overcome by applying homogenizing operations. The result is the homogeneous, federated schema.
3. *Derivation of external schemata*: This can be done in two further steps:
   (a) heterogenizing within the GIM and
   (b) transformation to a specific data model. The information captured can be reused here.

This work presents homogenizing operations in Chapter 3 and heterogenizing operations in Chapter 4 of the GIM. Additionally, in Section 4.2 the derivation of inheritance relationships is explained. The homogenizing and heterogenizing operations have to be inverse to each other. In other words, the same heterogeneity has to be capable of creating itself again, *after* the homogenization operations have been used, through applying the heterogenizing operations.

The same heterogeneity, which can be overcome by homogenizing operations has to be reproducible by heterogenizing operations. Furthermore, the set of homogenizing and heterogenizing operations has to be minimal and the operations have to be as orthogonal as possible.

Only the most important operations will be represented here. It seems to be impossible to prove that a given set of homogenizing operations will cover all possible conflicts. Many operations are based on ideas described in [BLN86].

The operations are not described formally. This will have to be done in future research. Graphical examples are used to clarify the operations. The examples will

be presented in a cartesian diagram. One dimension represents the extensional aspect whereas the other dimension serves to represent the intensional aspect of a class. A class itself can be depicted by a rectangle. References between two classes are illustrated by lines connecting the respective reference attributes. Integrity conditions are located below the extensions of the classes. Integrity conditions include cardinality constraints on references and values of class attributes, which are fixed to each object and necessary in order to restructure classes correctly. See also in Figure 2.1. The integrity condition "a5=10" assigns a fixed value to the class attribute a5.

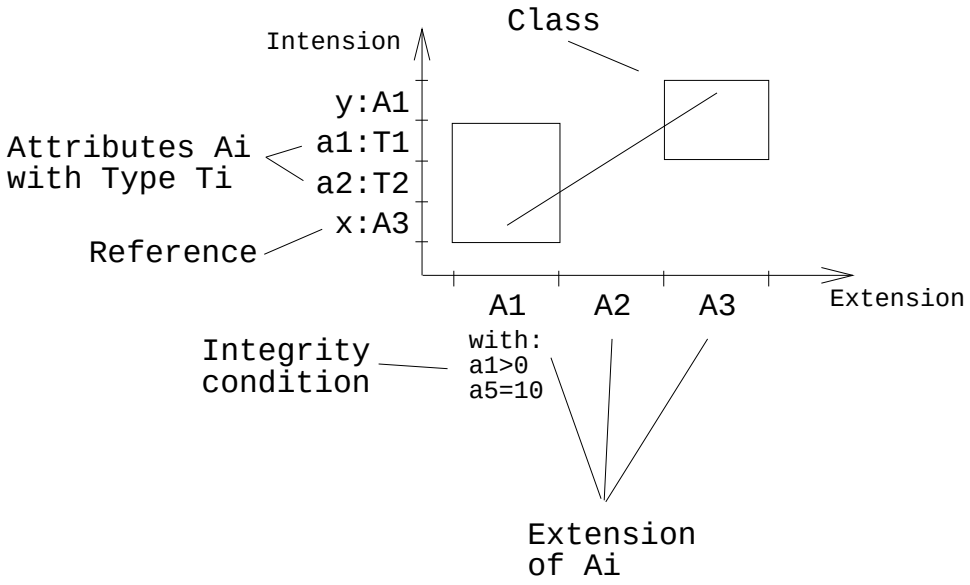

Figure 2.1: Diagram conventions

Chapter 3

Schema Homogenizing

Homogenizing of heterogeneous component schemata means the resolving of inter-schema conflicts. This work only considers conflicts between two schemata of equal weight. Conflicts between more schemata can be reduced to two schema conflicts [BLN86]. In the following sections the schemata are depicted in different diagrams. They can be merged into one schema after the extension-intension transformations on the schemata have been carried out.

The different operations are described in three steps:

1. precondition;
2. conflict description and
3. conflict resolution.

A missing precondition means no precondition.

The operations in the next sections are subdivided in operations concerning either the intensional aspect, the extensional aspect or both.

3.1 Homogenizing by Extension-Intension Transformation

This section describes two operations concerning transformations between intensional and extensional concepts of classes. These two operations resolve the *meta conflict* and the *structural conflict* in GIM:

- *Meta conflict*
  Conflict description: Objects of one class differ in values of a specific attribute (intensional) and the corresponding objects in the other schema differ in their class assignment (extensional). The attribute values in the first schema correspond to classes in the other schema. This conflict is referred to in [KLK91] as schematic discrepancies. [CL94] resolves this conflict in an object-oriented data model by using a query.
  There are two kinds of meta conflicts:

  - *injective:* There is a mapping from the domain of the datatype to the classes which is injective and maintains semantics. Such a mapping often occurs, if the datatype is relatively small, e.g. an enumeration or the boolean type. See the example in Figure 3.1.
  - *non-injective:* There is no injective mapping. For example, in one schema, persons can be distinguished by their occupations whereas in the other one by being an instance of the "Engineer" or the "Non-engineer" classes (see Figure 3.2).

  Conflict resolution: There are two ways to resolve this conflict. Either the attribute variant or the class variant has to be preferred. Classes in the class variant can have different intensions. The transformation of these classes to one class in the attribute variant would imply the problem of defining the intension of the class in the attribute variant without null values. Thus, the class of the attribute variant has to be transformed to the class variant. This can be done by creating corresponding classes and populating them with the respective objects obtained by select commands. The created classes have the same intension as the original class. In case of injectivity the respective attribute can be removed due to redundancy. Integrity conditions of the original class also have to be transformed properly. The values of the specific attribute in the attribute variant are transformed to values of class attributes of the newly created classes, e.g. the class attribute assignment "type=Book" in Figure 3.1 and "occupation=Engineer" in Figure 3.2.

- *Structural conflict*
  Conflict description: An attribute value of a class of one schema corresponds to an object in the other schema (see also [BL84, Dup94, LL95]). This conflict occurs due to different perceptions of the database designers. For example in Figure 3.3

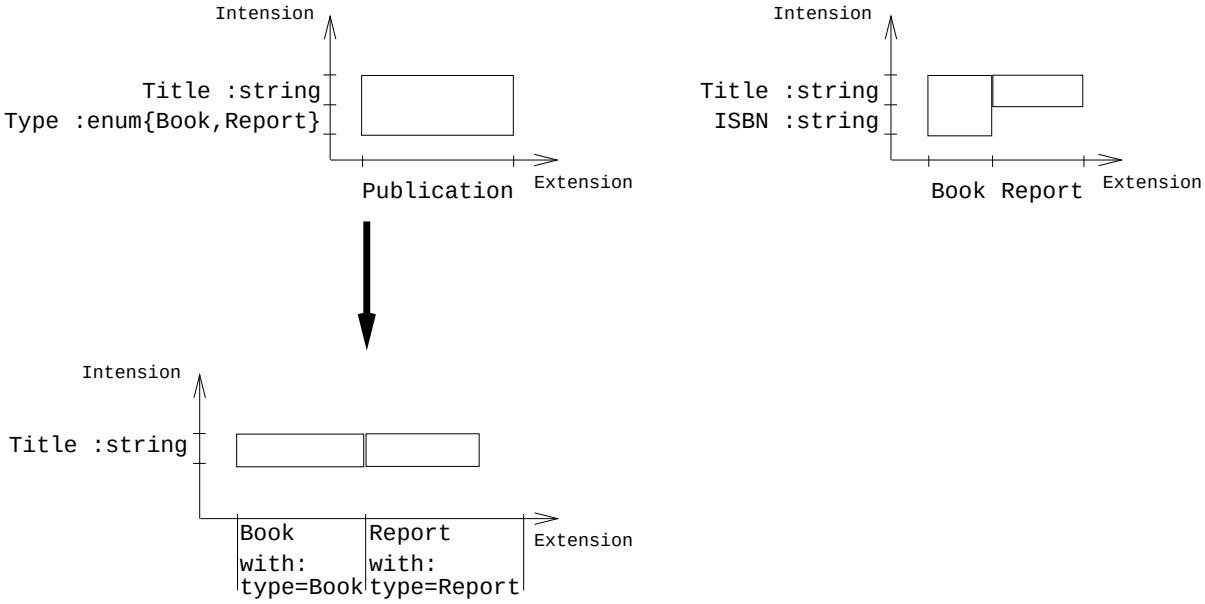

Figure 3.1: Injective meta conflict

the "Residence" of a person in the left schema is an attribute whereas the right schema contains the class "City". A value of attribute "Residence" corresponds to an object of the class "City".

Conflict resolution: There are two ways to resolve this conflict. The objects in the object variant which corresponds to the attribute values in the attribute variant can have more than one attribute. In the example in Figure 3.3 the class "City" has the additional attribute "Foundation". In order to avoid anomalies the object variant is preferred. The attribute variant has to be transformed to the object variant. That means creating a new class with an identifying attribute and transforming the attribute values to references to the newly created corresponding objects. The virtual objects also have to have object identifiers. In the example the attribute "Residence" becomes a reference attribute to the newly created class "City" which has the identifying attribute "Cityname". The transformation has to include the transformation of existing integrity conditions and the creation of two new integrity conditions which represent an existential dependency of the created objects from the referencing objects and the single-valued cardinality of the new reference.

All additional attributes in the schema containing the attribute variant which correspond to objects in the other schema, have to be transformed to reference attributes of the newly created objects.

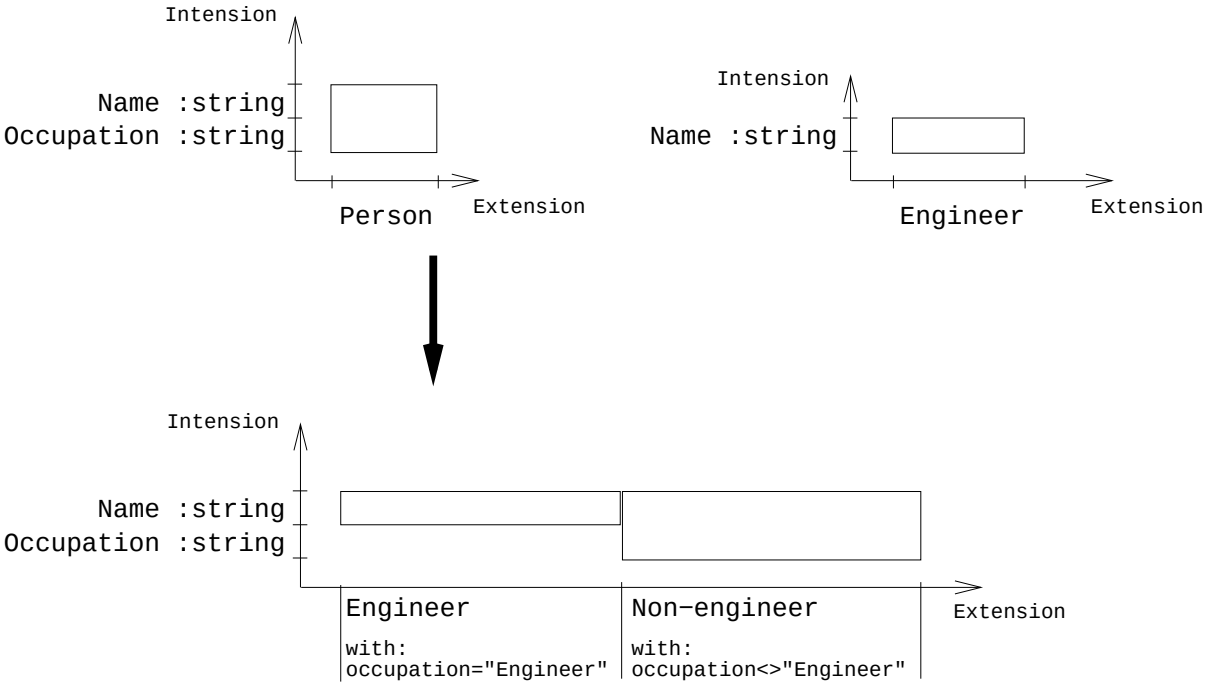

Figure 3.2: Non-injective meta conflict

3.2 Intensional Decomposition

In this section operations to resolve conflicts between the intensions of two corresponding classes irrespective of their extensions are described. The *attribute conflict* is a conflict between the domains of corresponding attributes of two corresponding classes whereas the *attribute set conflict* is a conflict between the attribute sets (intensions) of two corresponding classes. The intension of a class includes reference attributes.
Precondition: Before these operations can be carried out any meta conflict and any structural conflict has to be resolved.

- *Attribute conflict:*
  Precondition: Corresponding classes-attribute pairs which are semantically equivalent have to be provided by the designer of the federation.

  Conflict description: The datatypes of the corresponding attributes are different or the values of the attributes which are semantically equivalent are syntactically different. For the purposes of integration such corresponding attributes have to be represented virtually by non-redundant and integrated attributes with fixed datatypes. Such an attribute integration requires bidirectional mappings between the values of the starting datatypes and the values of the integrated attributes. Bidirectional mappings are neccessary to allow select and update operations on the integrated attributes. The mapping can also be the identity function. There are different ways to implement the mappings. A mapping can be implemented by

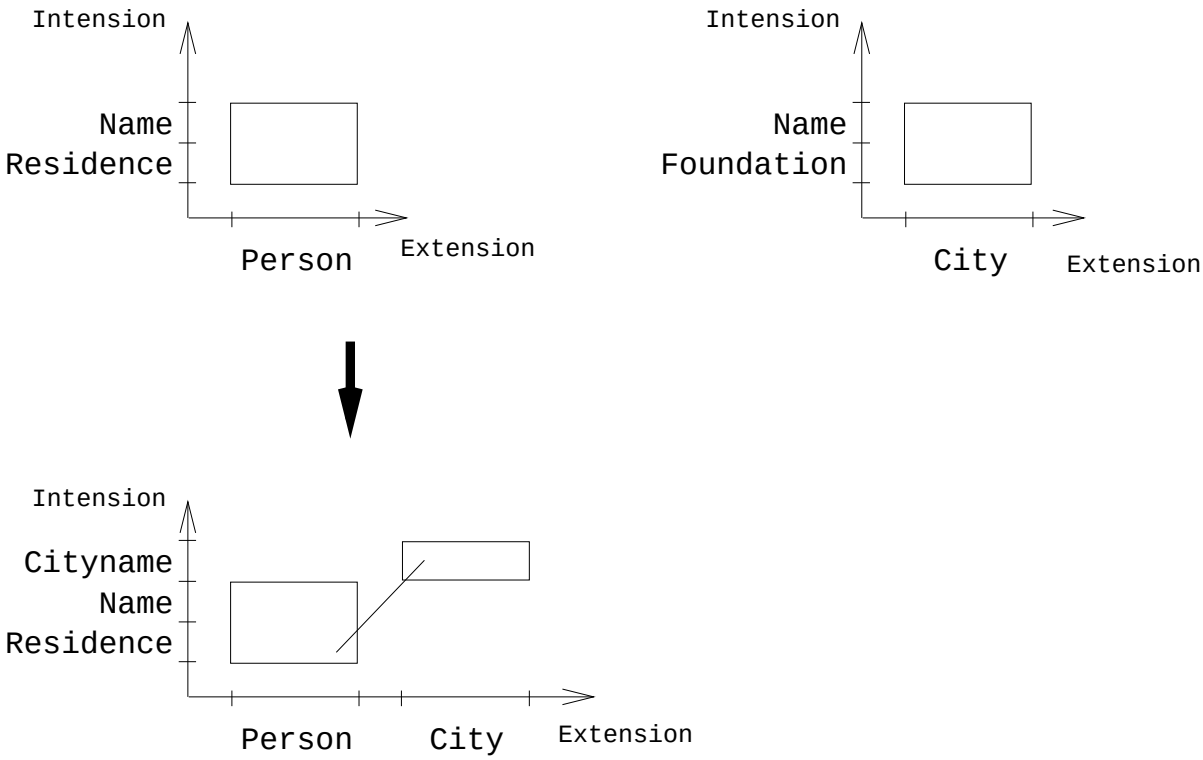

Figure 3.3: Structural conflict

two functions which compute operationally the values of the integrated attribute from the values of the starting attribute or vice versa. Another way to implement a mapping is the use of a lookup table which contains value pairs of the starting attribute and the integrated attribute (see also in [DeM89, YP95]). Lookup tables can be managed by the global database management system of the FDBS.
The conflicts between corresponding attributes are divided here into 1:1 conflicts and into conflicts which are conflicts between more than two corresponding attributes:

- *1:1 attribute conflicts*
  These kinds of conflicts are attribute conflicts between exactly two corresponding attributes. For example, in one schema the birthday of a person is represented by the attribute 'birth' of the datatype 'Date' and in the other schema it is represented by the attribute 'year_of_birth' of the datatype 'String(4)'. Regarding the possible mappings between the values of two corresponding attributes, the possible mappings to the integrated attribute can be divided into bijective, non-total (partial) and non-injective mappings.
  - *Bijective mapping*
    Conflict resolution: Bijective mappings are trivial and can be implemented directly.

* *Partial mapping*
  A mapping between two corresponding attributes is partial if there are not for all values of one attribute corresponding values of the other attribute.
  Conflict resolution: The datatype of the integrated attribute has to be chosen in a way that it encompasses all possible values of both starting attributes. The limits of the possible values of a starting attribute, which caused the partial mapping, have to be transformed to attribute constraints (integrity conditions) of the respective class.
  Figure 3.4 demonstrates the resolution of a partial attribute conflict. The datatype of the attribute 'name' is 'String[20]' in class 'Person' and 'String[40]' in class 'Insured'. The datatype of the integrated attribute is 'String[40]'. In class 'Person' this attribute is restricted by the attribute constraint $length(name) < 21$ of the class 'Person'.

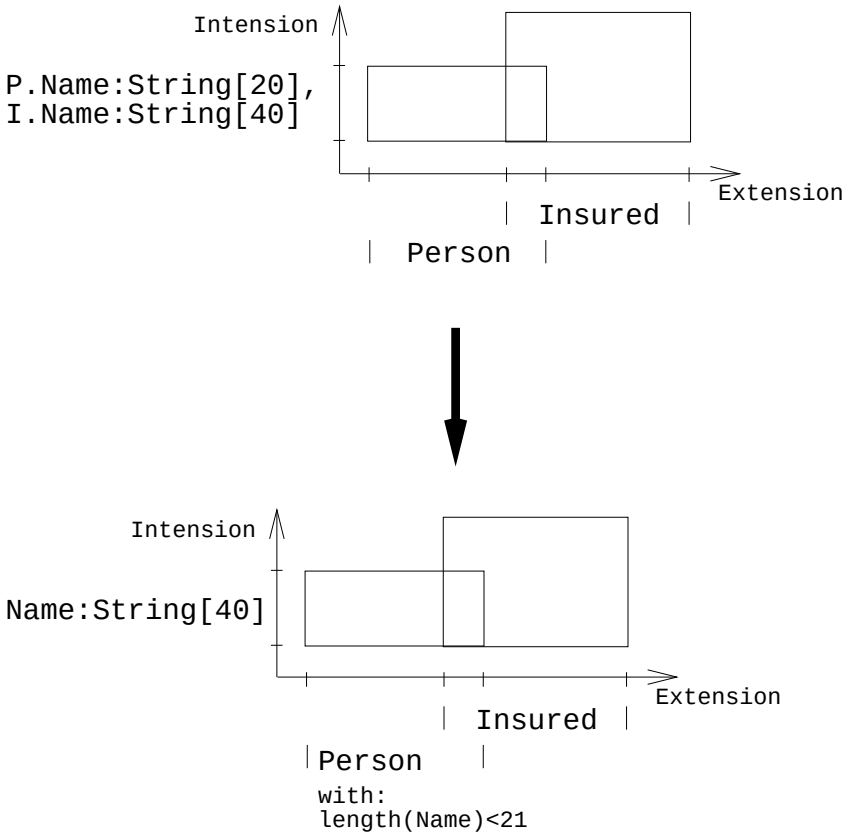

Figure 3.4: Partial attribute conflict

* *Non-injective mapping*
  A mapping between two corresponding attributes is non-injective if more than one value of one attribute is represented by only one value of the other attribute. This kind of attribute conflict is caused by attributes of different datatype precision. For example, the weight of a person can be

represented by the datatype 'Integer' or 'Float'. The float values 60.3 and 60.1 correspond to the integer value 60. Another example concerns dates. In one attribute a date is represented by combination of day, month and year and in the other attribute only by the year.
Conflict resolution: The attribute with more precision can be divided into attributes in such a way that there is an injective mapping between one resulting attribute and the attribute with less precision. Mappings between the other resulting attributes and the starting attribute with less precision are impossible.
Figure 3.5 exemplifies the non-injective attribute conflict and its resolution. The attribute 'I.birthday' is divided into the attribute 'year_of_birth', 'month_of_birth' and 'day_of_birth'. Now there is an injective mapping between the integrated attribute 'year_of_birth' and the attribute 'P.year_of_birth'.

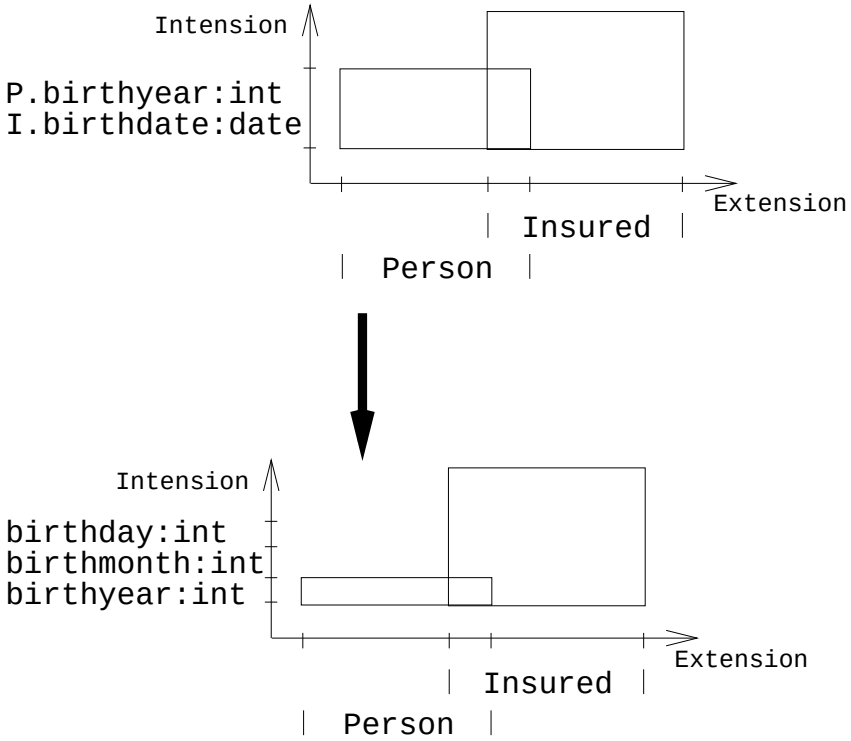

Figure 3.5: Injective attribute conflict

– *Attribute conflicts which are not 1:1*
This kind of attribute conflict is a conflict between more than one attribute of one class and one or more attributes of the other class. For example, in Figure 3.6 the full name of a person in one class is represented by the attribute 'name' of the datatype 'String[40]' whereas the name in the other class is represented by the two attributes 'first_name' and 'last_name' of the datatype 'String[20]'.

Conflict resolution: This conflict is a special case of the non-injective mapping conflict. In the example injective mappings between the attribute 'first_name' and the attribute 'name' of the other class and between the attribute 'last_name' and the attribute 'name' are impossible. Furthermore, there is a bijective mapping between the combination of the attributes 'first_name' and 'last_name' and the attribute of the other class.
An attribute conflict which is not 1:1 can be resolved by division of the attribute which contains more information in such a way that bijective mappings between any resulting attribute and the respective attribute of the other class arise. In our example the attribute which represents the full name has to be divided into two attributes, which represent the first name and the last name respectively.

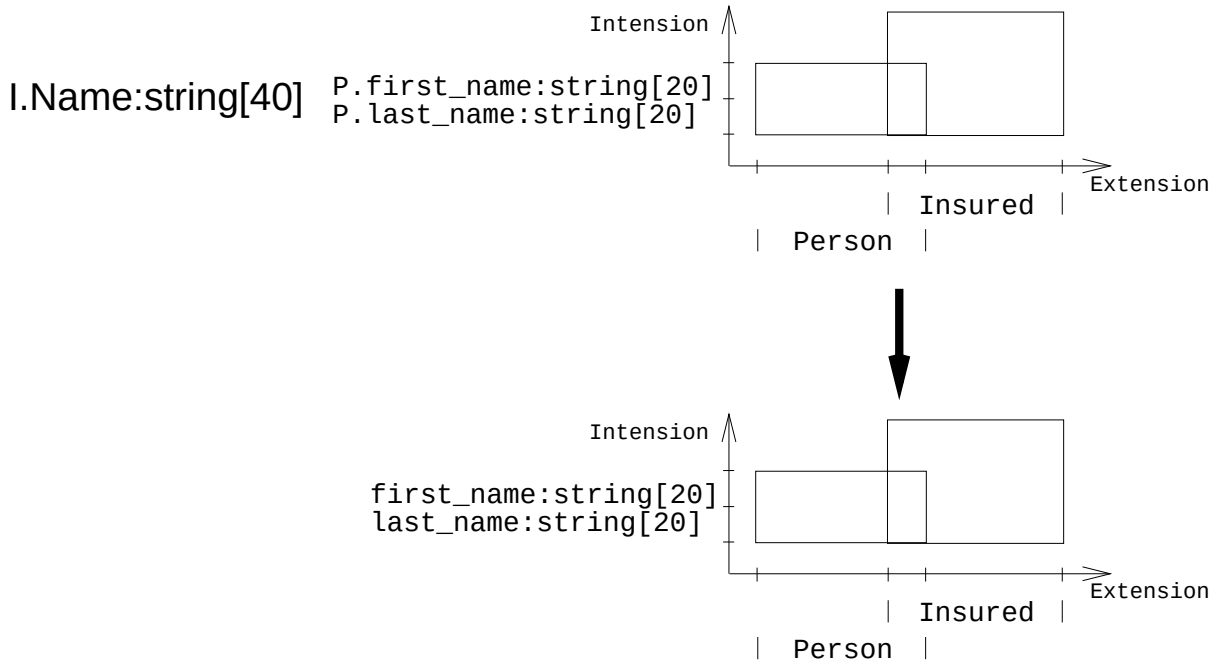

Figure 3.6: 1:n attribute conflict

- *Attribute set conflict*
  Precondition: The corresponding classes are known and the attribute conflicts have been resolved.
  Conflict description: Two corresponding classes have different attribute sets (intensions). The most general form of an attribute set conflict arises if the intensions overlap mutually. Both classes have common attributes and attributes which are attributes of only one class. In Figure 3.7 this conflict is demonstrated.
  Conflict resolution: The classes have to be divided horizontally into new classes in such a way that the intensions of the resulting classes are either identical or disjunctive. The respective integrity constraints have to be inherited by the resulting

classes.

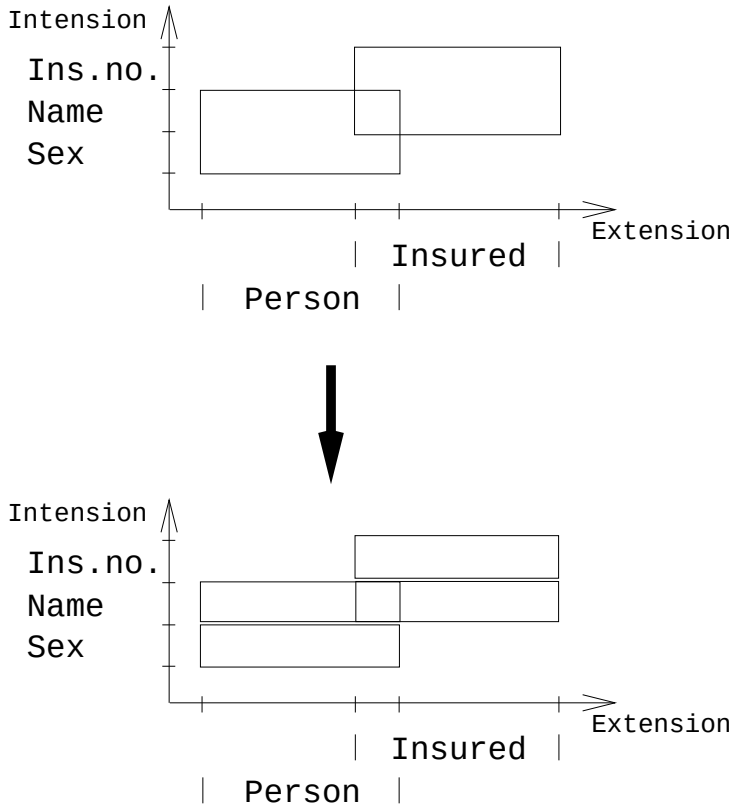

Figure 3.7: Attribute set conflict

3.3 Extensional Decomposition

In this section only extensional conflicts between two corresponding classes are considered. Extensional conflicts are conflicts between the possible populations of corresponding classes and are referred to as *semantical conflicts* here. The managing and the transformation of integrity constraints during the resolution of semantical conflicts are important topics. Some aspects of managing these constraints are described in the second part of this section.

Precondition: The corresponding classes are known and the intensional conflicts have been resolved.

- *Semantical conflict*
  Conflict description: The possible populations of two corresponding classes overlap mutually. Information about the same real-world entity can be stored in both corresponding classes which are managed by different database systems. In this context the term 'proxy-objects of the same real-world entity' is commonly used. In

[LSPR93, Pu91, ZHKF95] different ways of detecting and managing proxy-objects of the same real-world entity are described.

Conflict resolution: Most approaches resolve this conflict by inheritance, e.g. in [Dup94]. Our approach differs from the inheritance approaches. Overlapping class extensions are not allowed in GIM. This conflict is resolved by extensional decomposition. The result of the extensional decomposition are three classes which have disjunctive extensions. Two resulting classes contain those instances of the respective, starting classes which are not proxy-objects of same real-world entities and the third resulting class contains only proxy-objects of same real-world entities. Two proxy-objects of the same real-world entity are merged into one new object with its own object identifier (see also [SS95]).

Figure 3.8 demonstrates the extensional decomposition. An instance of the class 'Person' and an instance of the class 'Insured' can represent the same real-world person. After the decomposition the resulting class 'Person-Insured' contains information about persons which are not insured, the class 'Insured-Person' contains insured persons, which are not instances of the class 'Person' and the class 'Person&Insured' which contains insured persons.

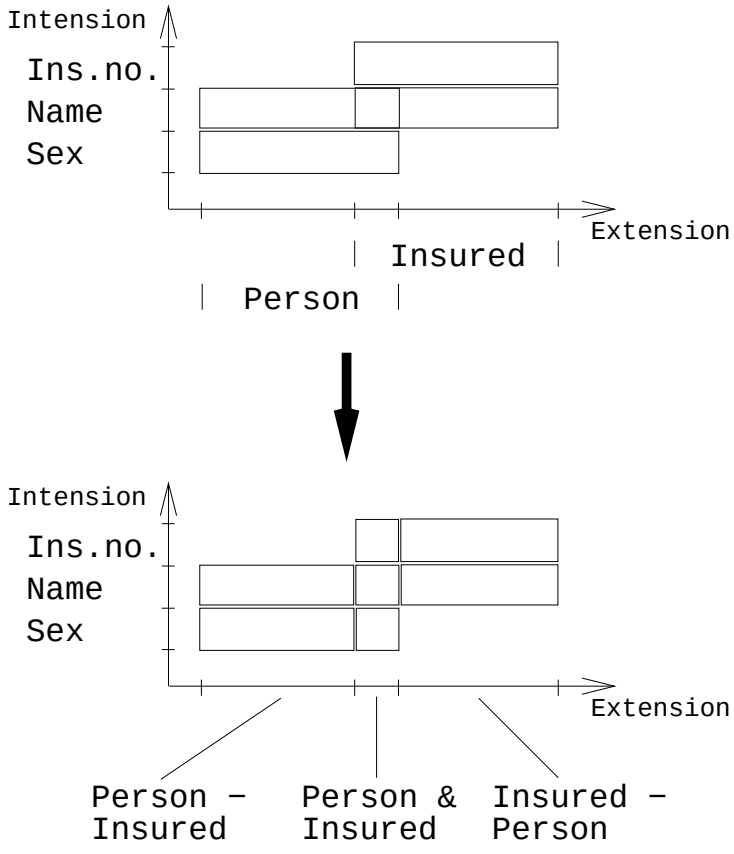

Figure 3.8: Semantical conflict

- *Integrity conflict*
  Conflict description: Corresponding classes with overlapping extensions can have different integrity constraints. The exensional decomposition has to consider conflicting integrity constraints. There are different kinds of integrity constraints:
  - *Statical versus dynamical integrity constraints:* Here we concentrate only on statical constraints.
  - *Intra-class versus inter-class integrity constraints:* Both types have to be taken into account.
  - *Object versus class integrity constraints:* An object integrity constraint is defined on the state of each object of a class whereas a class integrity constraint is defined on the state of all objects of a class, e.g. the key property. GIM supports both types.

  Conflict resolution: All integrity constraints survive the decomposition operation and are then defined on the resulting classes. Additionally, all class integrity constraints have to be transformed to inter-class integrity constraints. The resulting integrity constraints on the resulting class which contains the proxy-objects of same real-world entities are connected by the boolean operation 'and'.

  Figure 3.9 demonstrates the managing of integrity constraints during the decomposition process. The integrity constraints '$length(Name) < 41$' and '$length(Name) < 21$' are intra-class object integrity constraints and survive the decomposition process. After the decomposition the class 'P&I' has both constraints which restrict the length of the name. One constraint implies the other constraint. Thus, only the intra-class object integrity constraint '$length(name) < 21$' has to be checked. The constraint '$unique(Ins.no.)$' is a class intra-class constraint and is transformed to two intra-class class constraints on both resulting classes. Additionally, the new inter-class class integrity constraint '$unique(P\&I.Ins.no., I - P.Ins.no.)$' is created.

3.4 Homogenizing by Renaming

Conflicting names of classes and attributes are simple conflicts. They can be resolved by the renaming operation.
Precondition: Names of classes and attributes have been changed by the operations introduced in the previous sections. For example, the extensional decomposition creates new classes which need new names. Thus, conflicts resolutions which cause changes of names have to be carried out before the naming conflict can be resolved.
Conflict description: The operations introduced in the previous sections homogenize schemata without consideration of names of classes or attributes. Naming conflicts are different names of semantically equivalent classes or attributes (synonyms) or same names of semantically different classes or attributes (homonyms). The detection of

homonyms and synonyms can be supported by linguistic methods (described in more detail in [BH91]).
Conflict resolution: Naming conflicts are resolved by renaming or dropping synonyms. All synonym and homonym conflicts have to be resolved in such a way that there is a bijective mapping between classes and attributes, and names.
New attributes are generated by the resolution of intensional conflicts and new classes are generated by resolution of extensional conflicts. The designer has to decide whether one of the names of the starting attributes and starting classes respectively becomes the name of the new attribute and the new class respectively, or new names have to used. During the selection of right names changing scopes of existing names have to be considered.
For example, in Figure 3.8 the name 'Insured-Person' is generated. Usually, only persons can be insured. Thus, this name can be changed to the name 'Insured'. The resulting classes are shown in Figure 3.10.

Chapter 4

Schema Heterogenizing

The heterogenizing operations of this chapter enable the derivation of heterogeneous, external schemata from the federated schema. The same heterogeneity has to be capable of creating itself again, after the homogenizing operations have been used, through applying the heterogenizing operations. The heterogenizing operations in the next sections are not complete. We only show the inverse operations of the homogenizing operations.

- *Meta conflict:* The inverse operation of the homogenizing operation which prefers the class variant is the heterogenizing operation which prefers the attribute variant.
- *Structural conflict:* The inverse operation of the homogenizing operation which prefers the object variant is the heterogenizing operation which prefers the attribute variant.
- *Intensional conflict:* The inverse operations of the homogenizing operations which decompose classes intensionally are the homogenizing operations which compose classes intensionally.
- *Extensional conflict:* The inverse operations of the homogenizing operations which decompose classes extensionally are the homogenizing operations which compose

classes extensionally.

- *Naming conflicts:* Renaming of classes and attributes is simple and is not further detailled here.

Beside the reproduction of the same heterogeneity of the local schemata the heterogenizing operations have to enable the derivation of external schemata with arbitrary heterogeneity. Thus, homogenizing operations can also be used to derive external schemata.

The next Section 4.1 shows the operations which are inverse to the homogenizing operations of the meta and the structural conflict. Section 4.2 explains the composing operations which are inverse to the intensional and extensional decomposition operations. Attribute conflicts are not considered there. Additionally the derivation of inheritance hierarchies (not in GIM) will be described in Section 4.2.

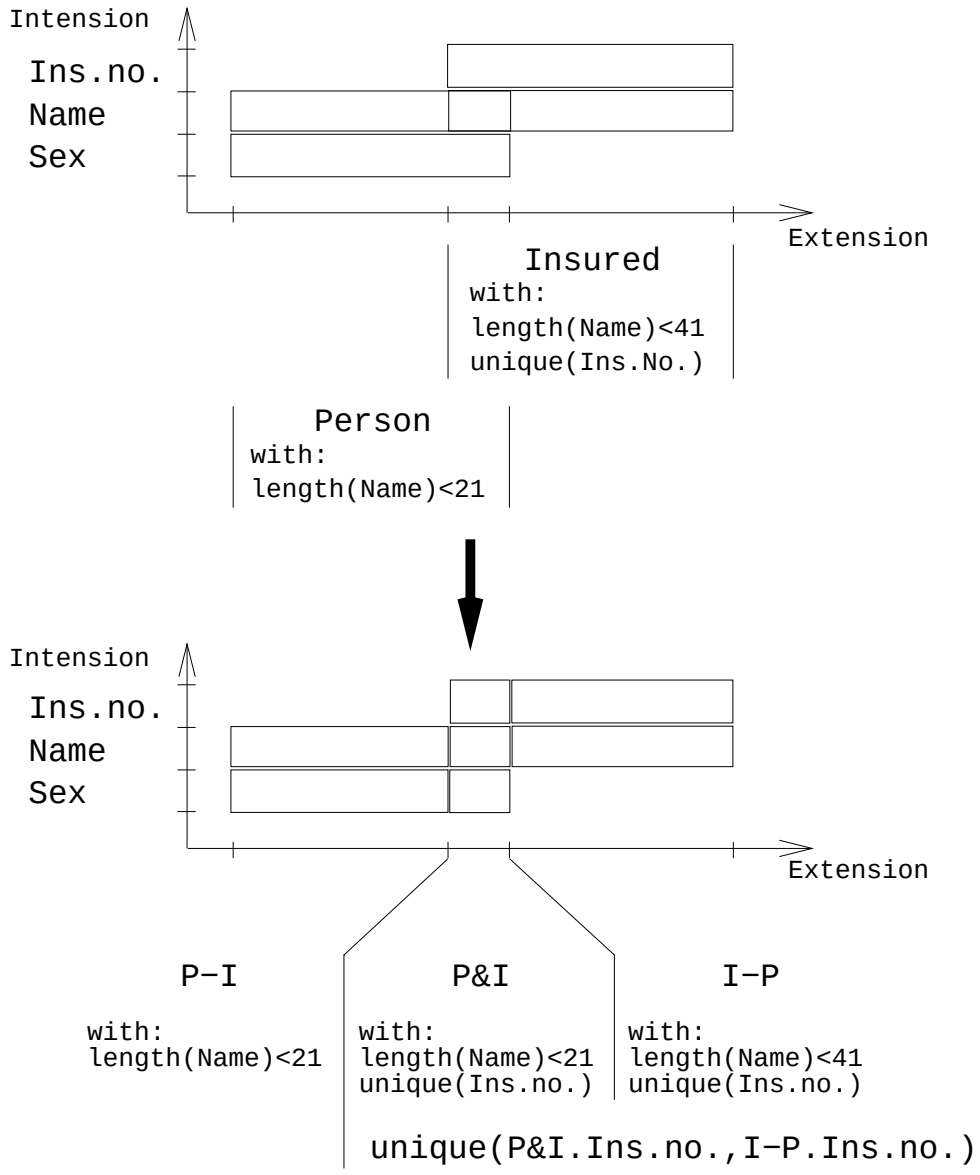

Figure 3.9: Integrity conflict

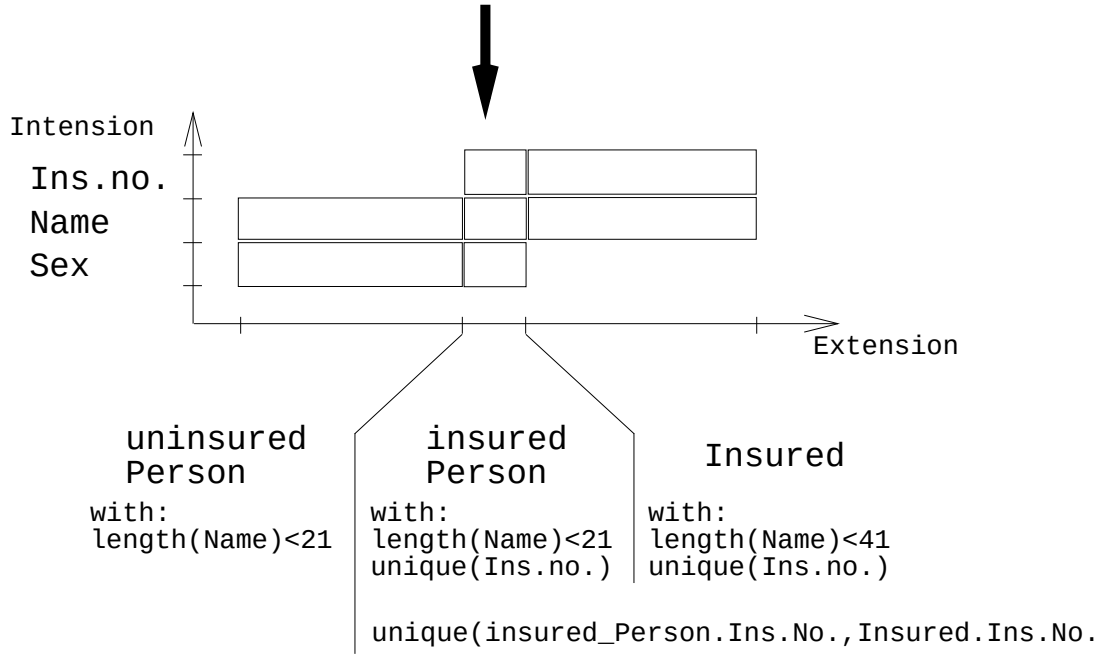

Figure 3.10: Naming conflict

4.1 Heterogenizing by Extension-Intension Transformation

In this section heterogenizing operations are described very shortly. They undo the homogenizing operations of Section 3.1.
Precondition: The federated schema is the start schema of any heterogenizing operations.

- *Meta conversion*
  View description: The information of the assignment of an object to a class has to be expressed by attribut values of a specific attribute (attribute variant). The starting classes have to have equivalent intensions. To each class the respective attribute values must be provided. This information can be either class attributes (which are generated by respective homogenizing operations) or existing attribute values (non-injective mappings).
  View derivation: The datatype of the pivot attribute has to be reconstructed from the class variants. After this, the extensions of the respective classes can be united into the class in the attribute variant and the respective values are assigned to the pivot attribute.

- *Structural conversion*
  View description: References to objects have to be transformed to attribute values (attribute variant). This operation is only possible if the referenced objects have only one identifying attribute. Otherwise information will be lost.
  View derivation: The respective attribute values have to be derived from the references. After this, the references can be removed. Many-valued references have to be represented by different attributes. The view derivation includes the transformation of integrity constraints.

4.2 Composing Classes

Different classes can be composed intensionally and extensionally. Due to the non-overlapping nature of the classes in GIM the composing operations are relatively simple. The intensional composition of two classes is feasible if both classes have equivalent extensions.
The extensional composition is more difficult than the intensional composition. Both classes have to have equivalent intensions. Otherwise, null values are generated by the composition operation. An important topic is the managing of different integrity constraints defined on the extensions of both starting classes. For example, in one class the integrity constraint '$length(name) < 21$' and in the other class the integrity constraint '$length(name) < 41$' is defined. The extensional composing of classes with different integrity conditions is only feasible with restrictions. There are two ways to manage such conflicting constraints:

- The integrity constraints are connected by the boolean operation 'and'. The extension is restricted to those objects which satisfy this constraint. Not all objects can be read. For example, objects cannot be read if the length of the value of attribute 'name' exceeds 20 characters.
- The integrity constraints are connected by the boolean operation 'or'. All objects can be read. Write or update operations can cause anomalies. Global applications are capable of changing the state of objects with regards to the composed integrity constraint but such objects cannot be stored in both base classes (as proxy-objects of same real-world entities). For example, an object cannot be the instance of the base class with the integrity constraint '$length(name) < 21$' if the length of the value of attribute 'name' is 25. One possible resolution is the declaration of read-only attributes. Otherwise anomalies must be accepted. Conflicting constraints on different attributes can even lead to situations in which new or updated objects cannot be stored in any base class.

The composing operations can be used to generate overlapping classes which represent an inheritance relationship between classes. The resulting schema is not conform to GIM. However such inheritance relationships are important if global applications need the federated schema which is based on a semantically rich data model. Valid inheritance relationships are crossing overlappings of the classes. The superclass encompasses the subclass extensionally and the subclass encompasses the superclass intensionally. Following these rule different inheritance hierarchies are often derivable.

Figure 4.1 exemplifies the derivation of an inheritance relationship from the classes in Figure 3.8. The class 'Person' is a superclass of the class 'insured_Person'.

Chapter 5

Conclusions and Outlook

In the previous chapters different homogenizing and heterogenizing operations were introduced. These operations are defined on schemata in the data model GIM. GIM is a semantically poor data model. The selection of a semantically poor data model makes the homogenizing and heterogenizing in federated environments easier. The homogenizing operations cover most known conflicts. Due to the homogenizing by decomposing and heterogenizing by composing, flexible schema transformation and integration is possible. A very important topic is the managing of integrity constraints. In Section 4.2

some restrictions of deriving valid external schemata are shown. The virtual integration of heterogeneous database systems is a very complex task. Different requirements are contradictory, e.g. transparency of federation, support of autonomy of component database systems and update operations on global level. In federated environments the right compromise between the requirements has to be found.

The operations and GIM were described only informally. Therefore, misunderstandings and errors are possible. In further work GIM and the homogenizing and heterogenizing operations will be described formally. Furthermore, the managing of conflicts which were not considered have to be investigated. For example, conflicts of different reference pathes between two classes are not explained here.

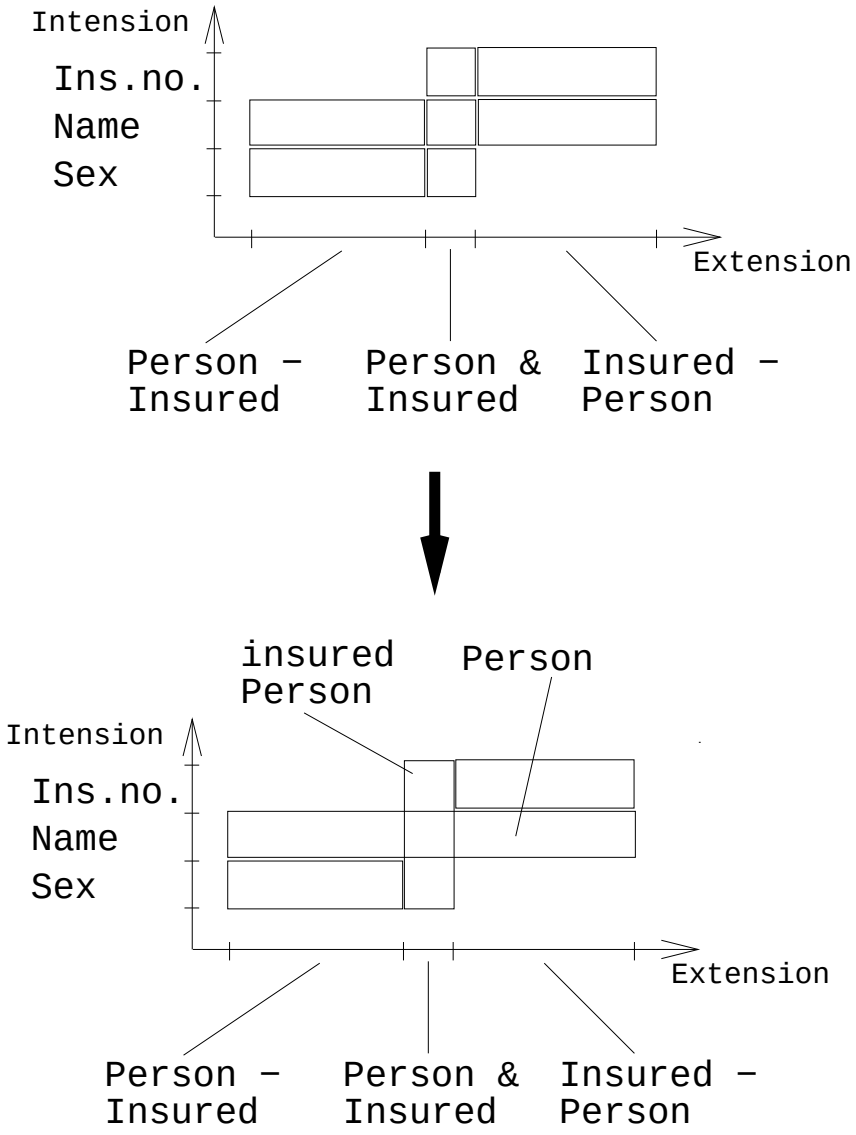

Figure 4.1: Inheritance relationship

Acknowledgements

Thanks to Michael Höding and Can Türker for many creative discussions. The author also has benefited from linguistic advice provided by Dietmar Schmitt, Andrea Diekmann and Maggie Dundee.

Bibliography

[BFHK94] R. Busse, P. Fankhauser, G Huck, and W. Klas. IRO-DB: An Object-Oriented Approach Towards Federated and Interoperable DBMS. In *Proc. of the Int. Workshop on Advances in Databases and Information Systems (ADBIS'94), Moscow, Russia*, pages 178–186. Russian Academy of Sciences, May 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.

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

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

[CHJ+96] S. Conrad, M. Höding, S. Janssen, G. Saake, I. Schmitt, and C. Türker. Integrity Constraints in Federated Database Design. Preprint Nr. 2, Fakultät für Informatik, Universität Magdeburg, April 1996.

[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.

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

[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.

[Här94] M. Härtig. *Objektorientierte Integration von autonomen Datenhaltungssystemen.* Dissertation, Verlag Dr. Kovač, Hamburg, 1994.

[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.

[Kim95] W. Kim, editor. *Modern Database Systems.* ACM Press, New York, 1995.

[KKM93] D. Keim, H.-P. Kriegel, and A. Miethsam. Integration of relational databases in a multidatabase system bases on schema enrichment. Technical report, Ludwig-Maximilians-Universität München, April 1993.

[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.

[LL95] M. L. Lee and T. W. Ling. Resolving Structural Conflicts in the Integration of Entity-Relationship Schemas. In M. Papazoglou, editor, *Proc. of the 14th Int. Conf. on Object-Oriented & Entity-Relationship Modelling (OOER'95), Gold Coast, Australia*, pages 424–433. LNCS 1021, Springer-Verlag, Berlin, December 1995.

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

[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.

[RBB+95] E. Radeke, R. Böttger, B. Burkert, Y. Engel, G. Kachel, S. Kolmschlag, and D. Nolte. Efendi: Federated Database System of Cadlab. In M. J. Carey and D. A. Schneider, editors, *Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, San Jose, CA, SIGMOD RECORD 24(2)*, page 481. ACM Press, June 1995.

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

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

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

[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.

[SS96a] I. Schmitt and G. Saake. Flexible Generation of Global Integrated Schemata using GIM. In S. Conrad, M. Höding, S. Janssen, and G. Saake, editors, *Kurzfassungen des Workshops "Föderierte Datenbanken", Magdeburg, 22.04–23.04.96*, pages 29–43. Bericht 96–01, Institut für Technische Informationssysteme, Universität Magdeburg, April 1996.

[SS96b] I. Schmitt and G. Saake. Integration of Inheritance Trees as Part of View Generation for Database Federations. In *Proc. of the 15th Int. Conf. on Conceptual Modelling (ER'96)*. LNCS, Springer-Verlag, Berlin, October 1996. *To appear.*

[SS96c] I. Schmitt and G. Saake. Schema Integration and View Generation by Resolving Intensional and Extensional Overlappings. In *Proc. of the 9th Int. Conf. on Parallel and Distributed Computing Systems (PDCS'96)*, September 1996. *To appear.*

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

[TS93] M. Tresch and M. Scholl. Schema Transformation Processors for Federated Objectbases. In S. C. Moon and H. Ikeda, editors, *Proc. of the 3rd Int. Conf. on Database Systems for Advanced Applications (DASFAA'93), Daejon, Korea*, pages 37–46. World Scientific Press, April 1993.

[YP95] J. Yang and M. Papazoglou. A Frame-Based Approach for Interoperation Support in Multidatabase Systems. In *Proc. of the 6th Int. Conf. on Database and Expert System Applications (DEXA'95), London, UK*, pages 48–57. Springer-Verlag, September 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.