# SCHEMA INTEGRATION FOR OBJECT ORIENTED DATABASE SYSTEMS

**Johann Eder and Heinz Frank**
Institut für Informatik
Universität Klagenfurt
Klagenfurt
Austria / Europe
email: {eder, heinz}@ifi.uni-klu.ac.at

## ABSTRACT

View integration is an important technique for developing software systems built upon large databases - independent which paradigm is used for realizing the database. We analyzed to which extent traditional view integration methods cover all aspects of object-oriented datamodels and found, in particular, that the integration of methods is hardly supported. We present a methodology for integrating schemas taking into account the statics as well as the dynamics of object oriented databases.

## 1. INTRODUCTION

The design of databases plays a crucial role in the development of large software systems. The ANSI/SPARC three level architecture (Tsichritzis and Kluge, 1978) supports a flexible and modular design of large application systems by providing mappings between the individual views of application programs or users on the data space (external models), the conceptual model representing the whole universe of discourse, and the internal model containing the physical data structures.

When we design databases for large software systems it is typically not possible to design the conceptual model at once, but it is much more feasible to first describe the universe of discourse from the viewpoint of parts of the system or of user groups resulting in a set of external models. In a second step, these different models are then integrated into one conceptual model. As people may have quite different - even eccentric - viewpoints, and the usage of the data in different parts of the system may suggest very different structures for the same data, this integration process is far from trivial.

For software systems which are built on several already existing databases (legacy systems) we face a similar problem - the conceptual models of the participating, maybe partly redundant, databases have to be integrated into one coherent conceptual framework, upon which the software system is then designed. Therefore, methods for assisting the view integration process are also helpful for the design of multidatabase systems or federated database systems (Kim et al., 1993 and Sheth and Larson, 1990).

Schema integration is not a new problem. It was subject of intensive research in the 80's, resulting in several well applicable integration methodologies mainly for the relational model and the Entity Relationship model (Chen, 1976), e. g. Ahmed et al., 1991, Batini and Lenzerini, 1984, Batini et al., 1986, Breitbart et al., 1986, Gotthard et al., 1992, Geller et al., 1992, Kim and Seo, 1991, Navathe et al., 1986, Schrefl, 1987, Sheth and Larson, 1990, Wiederhold, 1992, and Wiederhold et al., 1992. With the advent of object oriented databases with their richer set of modelling primitives we face the view integration problem again, although the notion of external models is poorly supported in current object oriented database systems. However, several approaches to reintroduce the three level architecture have already been proposed, e. g. Ahmed et al., 1991, Dobrovnik and Eder 1993a and Dobrovnik and Eder, 1993b.

The object oriented data models (Wegner, 1992) offer a richer set of concepts for representing the statics of the universe of discourse (types, classes, inheritance, etc.) than the relational model and, furthermore, also support the modelling of the dynamics (operation on the objects - methods). Therefore, they introduce several additional problems to the view integration process in the static domain as well as in the dynamic domain and the static - dynamic domain, as it is possible to design the same information as static property in one view and as dynamic (computed) property in another view. Because of the similarity between extended E-R-models and the static part of object oriented data models, solutions for problems like naming conflicts, structural conflicts, or data conflicts can easily be

adapted. Of course, a methodology for integrating object oriented views depends on the used data model. For defining our approach we choose the data model presented in Dobrovnik and Eder (1993a and 1993b). Nevertheless, the main concepts of our approach are applicable for other data models as well.

## 2. A CLASSIFICATION SCHEMA FOR VIEW INTEGRATION METHODS

### 2.1 Overview

In order to compare and analyze schema integration methods, we develop a classification schema which is based on Schrefl (1987) but extended to cover the concepts of object oriented data models.

The main steps in view integration are conflict analysis, merging, enrichment, and restructuring of schemas, although the partition of the whole process into steps and the naming of these steps are not homogeneous in the literature.

Generally, all view integration methodologies have to deal with representation conflicts, when the same information is designed with different features of the used data model, time-invariant connections between different component schemas, redundancies, and implicit semantic information between different component schemas, which have to be captured explicitly (schema enrichment).

According to that generalized problems we analyzed the integration methodologies with respect to:

- ♦ the used data model
- ♦ the proposed strategies to solve representation conflicts
- ♦ the approaches to handle redundancy removal
- ♦ the concepts to deal with schema enrichment

### 2.2 Representation Conflicts

Representation problems occur when real world information is designed differently in the component schemas which have to be integrated. They can be classified as naming conflicts, scaling conflicts and structural conflicts.

**Naming conflicts** originate from using equal terms for different real world concepts (called homonyms) or from different schema components representing the same real world concept under a different name (called synonyms). To solve naming conflicts

- ♦ the homonyms/synonyms in the component schemas are renamed
- ♦ new names in the integrated schema are introduced and mappings between the corresponding names in the component schemas are specified
- ♦ it is assumed that names in different schemas are only equal, if explicitly stated

**Scaling conflicts** arise if different scales are used for the same measure. For integration we use the finest scale and employ conversion functions to calculate the values for the external schemas.

**Structural conflicts** occur when the same information is represented with different constructs of the data model. Problems in this area can be found in

- ♦ aggregation, i.e. missing attributes or methods
- ♦ generalization, i.e. all other structural conflicts
- ♦ abstraction, i.e. differences in specialization and decomposition.

As the object oriented data models handle also dynamic aspects of the universe of discourse (beyond static aspects) representation conflicts occur when the same information is designed as static property in one view and as dynamic (computed) property in another view (**static-dynamic conflicts**). As an example suppose that the age of a person can be stored within an attribute 'age' or computed with an method on basis of the birth date.

### 2.3 Integration constraints

Integration constraints describe time invariant connections between component schemas and are necessary for detecting redundancy. We considered the following integration constraints in the analysis:

**object/class constraint** states that extensions of two classes are the same for all points of time

**attribute identity constraint** states that two attributes of different types are semantically equal. According to this also the values of two semantically equal attributes of two different objects which represent the same world entity must be the same.

**method identity constraints** hold when two methods of different types are semantically equal

**subclass constraint** states that all extensions of one class are contained in the extensions of another class. A special form of subclass constraint is the selection constraint, i.e. an extension of class B can be derived by a selection of a class A.

**subtype constraint** is similar to subclass constraint but refers to types, i.e. a type P is contained in another type T.

**mutual exclusion** constraint states that two classes have mutually exclusive sets of instances, although they might have the same type.

### 2.4 Redundancy removal

Integration constraints describe situations where information is stored redundantly in different schemas (with exception of the mutual exclusion constraint). According to the features of object oriented data models the following situations have to be analyzed:

**Redundant classes** can be removed, if an object/class constraint or a subclass constraint holds. In that case the classes and, therefore, also the types of the classes can be merged into one class and type. If a selection constraint holds, subclassing and subtyping can be applied.

**Redundant types** may be dropped, if a mutual exclusion constraint holds and the types are equivalent, as it is possible that two different classes have the same type.

**Redundant attributes** can be dropped, if an attribute identity constraint holds. In that case the attribute can be removed from the subtype. If an attribute can be derived from a composition of other attributes a new method can be introduced.

As object oriented data models propose dynamic aspects as well, **redundant methods** should also be removed (respectively merged). Of course, the process of determining equivalent methods is far from trivial. If methods are specified with a

formal specification, equivalent methods can be detected using a formal approach. More often methods are specified informal. In that case the designer has to analyze and to decide whether two methods are equivalent.

### 2.5 Schema enrichment/Interschema relationships

This is the process to capture implicit semantic relationships between different component schemas:

**type/class/object relationship:** Components of different user views can be related in various ways:

a) different types (probably partially equivalent type components) and different classes which are not disjoint.
Example: A class "CAROWNER" and a class "STUDENT", the types of both classes contain equivalent type components (e.g. Name or Address). Some extensions of both classes are equal, as there exist students who own a car. Possible solutions depend on the features of the used data model:

i) If the data model supports a role concept this feature should be used to handle such dependencies.

ii) Extract the common type components to a new type (say "PERSON") with the subtypes of "CAROWNER" and "STUDENT" (generalization). As some persons are contained probably in both classes, they can either be stored in both classes or another type as a subtype from both "CAROWNER" and "STUDENT" and a corresponding class have to be established. In the first case, two extensions of the different classes representing the same real world entity (e.g. the student Otto who owns a car) have different OID's (object identifier). In the second case all objects representing the same real word entity have the same OID.

b) different types with equivalent type components describing a common generic concept (e. g. PAPER and BOOK belong to the generic concept PUBLICATION). Schema enrichment is done by generalization.

c) equivalent types and disjoint classes, e. g. CUSTOMER and SELLER have equivalent types but no common extension. Schema enrichment is done by defining one type for both classes.

d) history related classes represent the same real world entity at different world times (e. g. classes APPLICANT and EMPLOYEE). They can be again organized in a generalization hierarchy. As none of the analyzed data models supports the time dimension we ignore this kind of relationship.

**component relationships:**

a) role-related attributes represent real world properties which describe probably a common generic property (e.g. University-Addr and Home-Addr). They are not semantically equivalent (that means their values may be different for one real world entity) but role-related. Only the types of the attributes can be integrated.

b) kind-related attributes represent real world properties which semantically overlap. They are integrated by first integrating the domains (e. g. define a generalization of the object classes) and then by combining the kind-related attributes to one attribute. For instance, in Austria, the social security number contains the birthdate of a person.

**method relationships:**

a) methods with identical semantics, i.e. methods computing the same result, although differences in the signature are possible (e.g. scaling)

b) role-related methods are methods which describe probably a common generic method

c) equivalent methods, which compute similar results but differ semantically, e.g. the domain or the semantic interpretation of the results is not the same.

d) overlapping methods: a method A contains (probably only partially) a method B. If possible the tasks of method A should be split into different methods.

e) general - special methods: a method A computes a more special task than a method B.

We present examples and solutions for method relationships in section 5.4.3.

## 3. COMPARISON OF SCHEMA INTEGRATION METHODOLOGIES

In this section we compare and analyze four different schema integration methods, two for extended E-R-models (Batini and Lenzerini, 1984 and Navathe et al., 1986) and two for object oriented models (Gotthard et al., 1992, and Geller et al., 1992) according to the classification scheme presented in section 2.

### 3.1 Navathe et al.: "Integrating User Views in Database Design"

Navathe's methodology for user view integration and global view integration is based on the Entity-Category Relationship data model (E-C-R) which includes also generalization hierarchies. Navathe et. al suggest the following integration phases:

**preintegration:** In the preintegration process naming conflicts between entities, attributes and relationships are resolved as well as scale mappings and the integration of similar entities (entities which are identical, contained, overlapping and disjoint). The approach does not cover structural differences between different user views but assumes that the same world concept is represented with the same construct in each user view.

**object integration:** To integrate two different object-classes a new superclass is created which contains the common attributes of the two integrated object-classes.

**relationship integration:** To integrate two relationships first the involved object-classes are integrated and then a new relation between the integrated object-classes is established with the original relationships as subrelationships.

### 3.2 Batini and Lenzerini: "A Methodology for Data Schema Integration in the Entity Relationship Model"

This methodology for view integration in the Entity Relationship Model provides three main phases: conflict analysis, merging and final enrichment and restructuring.

**Conflict analysis:** During this step representation conflicts, such as naming conflicts and structural conflicts are detected.

Naming conflicts are solved by renaming in the component schemas. Structural conflicts are handled by several restructuring steps where conflicting schema components are transformed (entity to relationship, relationship to entity and attribute to relationship). Conflicting integrity constraints are solved by choosing the more reliable one through the user.

**Merging:** Merging is done by a simple superimposition of common concepts. The result of this step is a three colored schema in order to distinguish concepts common to the two schemas and concepts inherited from a single schema.

**Enrichment and restructuring:** During schema enrichment and restructuring new relationships are used to increase the clarity and expressiveness of the new schema. Also redundancies within the integrated schema are removed.

### 3.3 Gotthard et. al: "System Guided View Integration for Object-Oriented Databases"

Gotthard et. al classify object oriented data models into two categories, structural object orientation and behavioral object orientation. The first paradigm emphasizes the arrangement of objects into clusters with establishing relationships between the objects. Behavioral object orientation focuses on objects as computational agents containing a set of procedures describing its external behavior and a private memory. Their methodology consists of the following three steps:

**Comparison of schemas:** During this phase all conflicts in representation of the same object in different schemas are detected (naming conflicts and structural conflicts) .

**Conforming of schemas:** The aim of this phase is to prepare different schemas for integration, that means to make them compatible for integration.

**Merging and restructuring:** The conforming schemas are merged by the concept of superimposition of common concepts and restructuring in order to get a complete, minimal and understandable global view.

### 3.4 Geller et. al: "Structural schema integration with full and partial correspondence using the dual model"

Geller et. al. make a difference between integration of already existing databases (schema integration) and the integration of different user (also application) views. They use the "Dual Model" which distinguishes between classes and their type (object type). They do not propose a methodology (in the sense of a high level algorithm) but concentrate on structural integration of types and classes. For that purpose they define several functions in order to establish some rules how to integrate classes and types. Geller et. al. describe:

**mappings between types and classes:** Conditions to find mappings between one object type to one class are defined as well as mappings from one object type to several classes.

**formal conditions for structural integration:** Structural integration between two sets of classes is possible, if there exists a correspondence between the two sets such that for every corresponding class from the sets a common object type can be constructed. Two cases of correspondence occur, full structural correspondence and partial structural correspondence. To construct a common object type for two corresponding classes a full structural equivalence between these classes , i.e. between their sets of properties must exists. In practical integration the rules for structural integration are also applicable to schemas differing only in view properties.

### 3.5 Summary

An overview of our results is given in appendix 1. For solving structural conflicts appropriate strategies were presented. However, only Navathe et al. (1986) deals with scaling conflicts by the introduction of conversion functions and conversion tables. Several integration conflicts are not covered by the approaches as none of them distinguishes between class and type integration explicitly.

We saw that the dynamic aspects of object oriented data models are not supported. Only Geller et al. (1992) shows an approach to solve method conflicts at signature level but cannot handle the semantics of the methods. None of the inspected methodologies are aware of conflicts in the static - dynamic domain.

Our methodology is based on the approaches sketched above. However, we aimed at overcoming the deficiencies of these methods. So our methodology detects and solves all conflicts detected and solved by these other methods but furthermore it distinguishes between class and type integration explicitly, detects and solves method conflicts at signature level and in the semantics of the methods as well as conflicts arising in the static - dynamic domain.

## 4. A SKETCH OF THE UNDERLYING DATA MODEL

In this section we briefly describe the data model on which our considerations are based (Dobrovnik and Eder, 1993a and 1993b). A schema in our data model consists of type definitions and class definitions. Like in other approaches we distinguish between types and classes. Types represent intensional information, while classes denote extensional information. So types describe the structure of objects and values, in particular, which components (instance variables) they consist of, which methods can be applied to them, and the signature as well as the implementation of methods.

Types are defined in the context of an inheritance lattice for structural inheritance. If we define a type t to be subtype of a type s, we mean, that the type t inherits all the structural information (components, signature and implementation of methods) from its supertype. In the definition of the subtype we can add additional components and additional methods. Components and methods inherited from the supertype can be override whereby we assume that the inheritance lattice forms a subtype hierarchy in the sense of covariant subtyping, i.e. wherever we can use an object of a certain type we also can use an object of one of its subtypes.

We define classes to be object containers, which can include objects compatible with a ground type. The instance of a class is the object set of the class. There may be no class, exactly one class or many classes of a certain type. So, there is no class automatically generated when a type is defined. Operators on classes are provided to include an object in a class,

to remove it from the class, to test if an object is in a class and to iterate over the object set of the class.

Classes can be arranged in a class hierarchy. Subclassing means subsetting the object set of a class. An object in a certain class C is also in all superclasses of C (instance inheritance). But additionally, an object may be in any number of unrelated classes if the ground type of the class is compatible with the object's type. The membership of an object to a class can be made explicit by means of the operators mentioned above. The second possibility to define the object set of a class is a predicate associated with the class. Predicative subclassing is a powerful means of automatically (re-)classifying objects.

In Dobrovnik and Eder (1993a and 1993b) a view definition concept has been developed. The possibility to define views is important for building large software systems as it increases modularity and by logical data independence it reduces maintenance due to schema evolution. An important aspect of view integration is, therefore, how the component views can be derived from the conceptual schema after the integration. However, this aspect is beyond the scope of this paper.

## 5. INTEGRATING OBJECT ORIENTED VIEWS

### 5.1 Overview

For our approach we define three main phases for schema integration, the preintegration phase, the class integration phase and the type integration phase. Like every other methodology, we do not present an algorithm in the sense of stepwise actions rather as a way of thinking, as these phases are overlapping. Especially the class and type integration are processes depending on each other, because it is not possible to integrate classes without integrating their types and vice versa. We propose to begin at class level as it is easier to find semantic relationships between classes than detecting structural equivalent types. Nevertheless, our approach leads a way through the difficulties of schema integration as we show the main operations handling such difficulties.

Now we give a brief overview of our approach, divided into the main phases and the main steps within each phase.

**Preintegration:** General conflict analysis according to types, classes, type components and methods in order to detect obvious conflict situations.

**Class Integration:** Semantic analysis, beginning at class level (partially parallel also at type level) in order to find relationships between classes, which can be classified as:

- ♦ not disjoint classes: superclass integration, type integration, class integration, subclass integration
- ♦ semantically equal but disjoint classes: superclass integration, type integration
- ♦ redundant classes: superclass integration, type integration, class integration, subclass integration

**Type Integration:** Structural analysis in order to find equivalent types.

- ♦ type integration
  - ⋆ solve structural conflicts
  - ⋆ integrate supertypes -> type integration
  - ⋆ integrate type components -> type component integration
  - ⋆ integrate methods -> method integration
  - ⋆ integrate static - dynamic domains -> static - dynamic integration
  - ⋆ type merging
  - ⋆ integrate subtypes -> type integration
- ♦ integrate corresponding classes -> class integration

As an example we choose to model two different views of a car dealer, the view of the selling office and the view of the service office (figure 1).

In the following sections we refer to this example in figure 1 to show the main steps in object oriented view integration. Boxes represent types, cycles are classes. Each type (with exception of AddressT) has at least one class. Type hierarchies are represented with an arrow whereas class hierarchies are represented with a dashed arrow line. The basic data types are defined as I for integer, S for string and D for date. Due to the lack of space we can only present a small example and had to skip the methods.

### 5.2 Preintegration Phase

During the preintegration phase we propose to do a naming, structural and scaling conflict analysis to solve obvious problems. In the example, both types, PersonT and AdressT contain an attribute describing the zip code of a city but named differently, zip and zipcode. These attributes can be renamed easily. Also a structural conflict occurs as both have different types. We have to come to terms about the type of these attributes in the conceptual model, say defining both as an integer.

Searching and solving such conflicts should be done very carefully as a lot of useful information emerges during this process. This information can be used within the class integration phase to find relationships between classes of the different schemas. The types EmployeeT from the service view and StaffT from the sales view are obvious semantically equal because they describe both members of the modelled organization, although their types seem to be very different. These types and the corresponding classes, therefore, seem to be good candidates for integration.

The result of the preintegration phases is not only a set of schemas better prepared for the following integration phases but also information on how to continue.

### 5.3 Class Integration

During class integration the classes have to be semantically analyzed to find relationships between them. Of course, it is difficult to find such relationships but there are some reference points to detect them, e. g.

♦ extensions of the classes, EmployeeC and StaffC contain both members of the organization such as salesmen, secretaries

♦ similar names of the classes are a possible way to find relationships

♦ the information which was collected during the preintegration phase

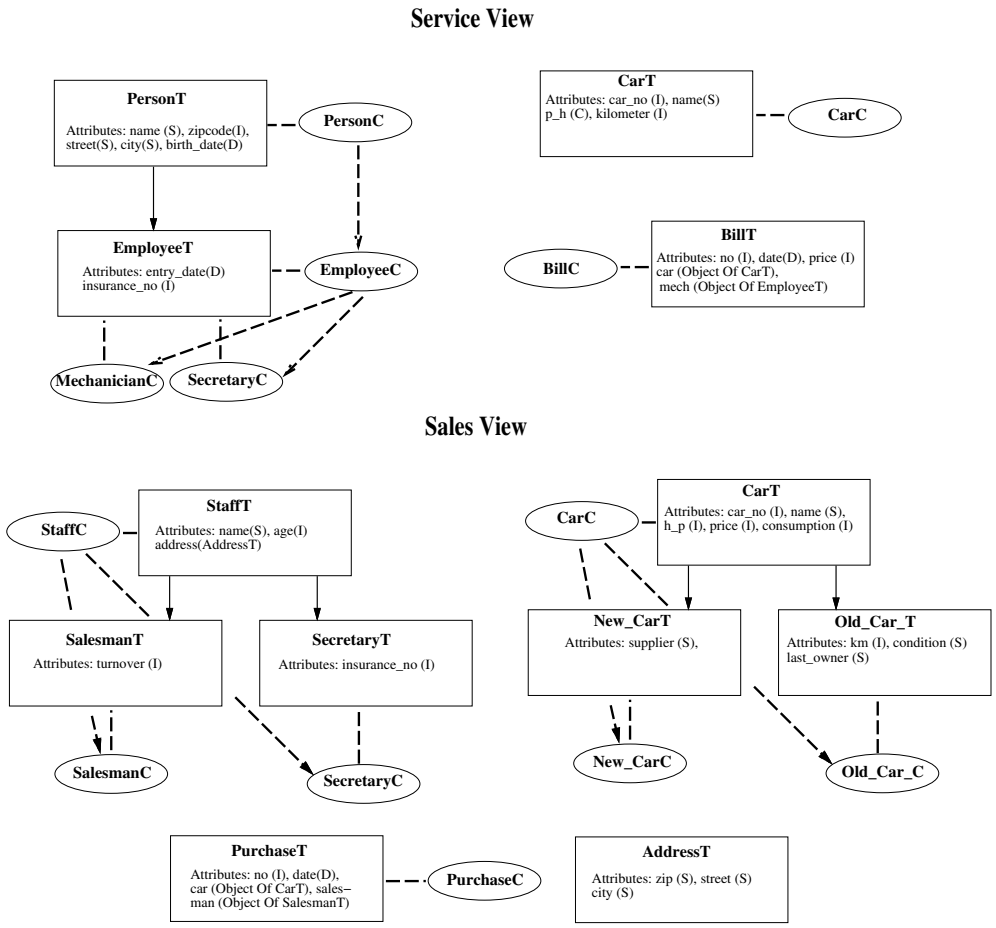


**Figure 1**: Graphical representation of the service view and the sales view

♦ a domain model of the organization (Eder and Rossak, 1992)

Relationships between classes can be classified as follows:

♦ semantically equal classes with partial different types which are not disjoint, e.g. CarC of service view and CarC of sales view

♦ semantically similar classes which are disjoint, e.g. MechanicC of service view and SalesmanC of sales view

♦ redundant classes (but probably different in types), e.g. the secretary classes of both schemas, which contain equal sets of objects

After having detected a relationship between two classes naming conflicts must be solved (if this has not yet been done). According to the above classification different solution strategies are usable.

**semantically equal, not disjoint classes**: By this we mean classes where the intersection of the extensions is not empty. In our example the classes StaffC of the sales view and EmployeeC of the service view are candidates for such an integration. To integrate such classes the following steps are necessary:

♦ integrate their superclasses. In the example first the PersonC as superclass from EmployeeC has to be restructured (as StaffC has no superclass).

♦ integrate the different class types, i.e. the EmployeeT and the StaffT of our example

♦ integrate the involved classes, by establishing a common class hierarchy

♦ integrate their subclasses, in our example the classes MechanicianC and SecretaryC of the service view and SalesmanC and SecretaryC of the sales view.

**semantically similar, but disjoint classes**: Semantically equal classes with an empty intersection of their extensions can also be integrated. Such classes describe equivalent real world entities but none of them can be in both classes. We refer to the classes SalesmanC and MechanicC in our example. Both contain members of the modeled enterprise but a salesman can not be a mechanic and vice versa. To integrate such classes first their types have to be integrated and, second, either two classes of the adapted type are defined or one class is established, if the classes can be derived by a selection (see selection constraint in section 2.3).

**redundant classes**: Redundant classes are classes representing the same set of real world entities, in our example the two classes SecretaryC are candidates for such an integration. After solving a possible naming conflict (if this has

not yet been done in the preintegration phase) the following steps are necessary:

♦ integrate their superclasses. According to our example the classes PersonC and StaffC have to be integrated. Note that the superclass integration might just be done during the integration of semantically equal classes with a common set of extensions.

♦ integrate the different class types, e.g. EmployeeT and SecretaryT

♦ integrate the two classes, i.e. one class is dropped

♦ integrate the subclasses

As we have seen not all possible steps are necessary to integrate classes. Because of the strong recursive character of our approach some of the steps are probably already made (e.g. superclass integration or naming conflicts).

### 5.4 Type integration

Within the type integration phase the structure of the involved types is analyzed in order to find a common type agreement between the schemas. For a more detailed discussion of structural integration we refer to Geller et al. (1992), who show how types with full structural correspondence and with partial structural correspondence can be integrated. Here we present only the main approaches to integrate class types.

Suppose two types have to be integrated, e. g. because a relationship between the corresponding classes exists, then the following steps are necessary:

♦ handle naming conflicts

♦ integrate the supertypes

♦ solve structural conflicts with respect to type components, methods and static/dynamics conflicts

♦ merge types to obtain a common type hierarchy

♦ integrate the subtypes

♦ integrate the corresponding classes with the class integration step

Having found a common type hierarchy also the corresponding classes must be integrated.

**5.4.1 Solving structural conflicts.** During the structural analysis step a common type agreement between the integrated types must be found. To obtain a common type hierarchy all parts of the types have to be integrated. Therefore, we distinguish:

♦ **type component integration**. By type components we mean the static part of a type. A type component may be a (complex) value or a reference to other objects.

♦ **method integration.** Several conflicts with respect to methods can occur, such as different, overlapping or redundant methods. The detection of equivalent methods is far from trivial. Formal approaches (formal specification methods) are appropriate as well as a semantic evaluation through the developer or analyzer.

♦ **static/dynamic integration**. For a definition of static/dynamic integration we refer to section 2.2.

**5.4.2 Type component integration.** For the integration of values and object references we refer to Geller et al. (1992), Navathe et al. (1986) and Kim and Seo (1991), where algorithms and approaches are presented to deal with naming conflicts, data type conflicts and scaling conflicts.

As an illustration remember the zip code example presented in the preintegration phase. The attribute address of StaffT and the attributes zipcode, street and city of the PersonT are equal and must be integrated, too. One possible solution is to define a type AdressT for the conceptual model. PurchaseT and BillT are similar types and should be integrated as they describe similar real world objects. The referenced object of both types (e.g. an object of SalesmanT in PurchaseT and an object EmployeeT in BillT) are similar and the classes SalesmanC and EmployeeC their types SalesmanT and EmployeeT have to be integrated.

**5.4.3 Method integration.** The integration of different methods is a very difficult part of object oriented schema integration. Beyond the static description of methods (i. e. the signature) also the semantics of methods have to be integrated. The conflicts at signature level, such as naming conflicts, data type conflicts, scaling conflicts, are similar to those in type component integration and, therefore, the same detection and solution strategies are applicable. For the more difficult semantic integration an informal approach by analyzing the method operation as well as a formal way is appropriate to detect relationships between methods. Such relationships can be classified:

♦ methods with identical semantics: the conceptual model contains only one method

♦ role-related methods, describing a common generic method, e.g. methods for computing the age and the employment time of a person, as they both compute the time between a given date and today. The conceptual model contains only that generic method.

♦ equivalent methods, which compute similar results but differ semantically. E.g. the consumption of a car is computed in continental Europe as liters per hundred kilometers but as miles per gallon in the USA. Both methods are equivalent although the results are semantically different. The introduction of conversion functions or method overriding in the definition of the external schemas are appropriate solution strategies.

♦ overlapping methods: a method from type A contains (probably only partially) one or more methods from the equivalent type B, e.g. a method computing the sum of a bill including taxes and one without taxes. Task splitting or explicit method redundancy solve such conflicts.

♦ general - special methods: a method from type A performs a more special task than a method from type B. Although the detection of a "method hierarchy" is a very hard problem, a schema integration methodology should take care about it. Method overriding within the type hierarchy is a possible solution. For example a method computing the area of a rectangle is much more general than one computing the area of a square.

A detailed discussion of method integration and finding relationships between methods including the comparison of formal specifications is subject of ongoing research.

### 5.4.4 Static/dynamic integration

A static/dynamic conflict occurs when the same information is stored as a static attribute in one schema and as a computed attribute (method) in a different schema. In our example the attribute age of StaffT and the method age of PersonT (not shown in figure 1) represent such a conflict situation. As solution strategies for such kinds of conflicts are possible:

- ♦ drop either the attribute or the method
- ♦ store the information redundantly (as attribute and as method)

### 5.4.5 Type merging

Types are merged after all structural conflicts are solved, i.e. a single type hierarchy is formed. The methodologies for structural type integration, specialization, generalization and aggregation are well described in the literature, especially we refer to Geller et al. (1992).

## 6. CONCLUSION AND FURTHER RESEARCH

We presented a classification for the problems or conflicts which have to be covered by schema integration methods for object-oriented databases. An analysis of such techniques in the literature showed that, in particular, the integration of methods and the distinction between types and classes are hardly supported. We presented a framework for a schema integration methodology which is tailored towards full object oriented datamodels. The integration of formal and semiformal techniques for the comparison of methods is subject of further research, as well as the application of domain modeling for the search of similarities between schemas.

## REFERENCES


Ahmed, R., De Smedt, P., Du, W., Kent, W., Ketabchi, M., Litwin, W., Raffi, A. and Shan, M.: "The Pegasus Heterogeneous Multidatabase System", IEEE Computers, December 1991, pp. 19-27

Batini, C. and Lenzerini, M.: "A Methodology for Data Schema Integration in the Entity Relationship Model", IEEE Transactions on Software Engineering, November 1984, pp. 650-664

Batini, C., Lenzerini, M. and Navathe, S.: "A Comparative Analysis of Methodologies for Database Schema Integration", ACM Computer Surveys, December 1986, pp. 323-364

Breitbart, Y., Olson, P. and Thompson, G.: "Database Integration in a Heterogeneous Database System", in Proceedings of the International Conference on Data Engineering, IEEE, 1986, pp. 301-310

Chen, P.: "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, March 1976, pp. 9-36

Dobrovnik, M. and Eder, J.: "View Concepts for Object-Oriented Databases", in Proceedings of the 4th International Symposium on Systems Research, Informatics and Cybernetics, Baden Baden, 1993a

Dobrovnik, M. and Eder, J.: "A Concept of Type Derivation for Object-Oriented Database Systems", in Proceedings of the Eight International Symposium on Computer and Information Sciences (ISCIS VIII), L. Gün et. al. (eds.), Istanbul, 1993b

Eder, J. and Rossak, W.: "Using a Data-Oriented Approach to Decide on Similarity of Objects During Domain Analysis" in: R. Trappl (ed.): "Cybernetics and Systems Research '92, Vol. 1, World Scientific, 1992, pp. 81 - 88

Geller, J., Perl, Y., Neuhold, E. and Sheth, A.:"Structural Schema Integration with full and partial correspondence using the Dual Model", Information Systems, Vol. 17, No. 6, 1992, pp. 443-464

Gotthard, W., Lockemann, P. and Neufeld, A.: "System-Guided View Integration for Object-Oriented Databases", IEEE Transactions on Knowledge and Data Engineering, February 1992, pp. 1-22

Kim, W., Choi, I., Gala, S. and Scheevel, M.: "On Resolving Schematic Heterogeneity in Multidatabase Systems", Distributed and Parallel Databases, July 1993, pp. 251-279

Kim, W. and Seo, J.: "Classifying Schematic and Data Heterogeneity in Multidatabase Systems", IEEE Computer, December 1991, pp. 12-18

Navathe, S., Elmasri, R. and Larson, J.: "Integrating User Views in Database Design", IEEE Computers, January 1986, pp. 185-197

Schrefl, M.: "A Comparative Analysis of View Integration Methodologies", in R. Wagner, R. Traunmüller, H. Mayr (eds.): "Informationsbedarfsermittlung und -analyse für den Entwurf von Informationssystemen", Fachtagung EMISA, 1987, pp. 119-136

Sheth, A. and Larson, J.: "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, September 1990, pp. 183-235

Tsichritzis, D. and Klug, A. (eds.): "The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Database Management Systems", Information Systems 3 (1978). AFIPS Press

Wegner, P.: "Dimensions of Object-Oriented Modeling", IEEE Computers, October 1992, pp. 12-20

Wiederhold, G.: "Mediators in the Architecture of Future Information Systems", IEEE Computers, March 1992, pp. 38-49

Wiederhold, G., Wegner, P. and Ceri, S.: "Toward Megaprogramming", Communications of ACM, November 1992, pp. 89-99

## Appendix 1: Comparison Table

| | *Navathe* | *Batini* | *Gotthard* | *Geller* | *Eder* |
|---|---|---|---|---|---|
| *Data Model* | E-C-R | E-R | CERM | Dual Model | OODB |
| *Representation* | informal | informal | informal | informal | informal |
| *view integration* | + | + | + | + | + |
| *schema integr.* | + | - | + | - | + |
| *class/type integ.* | - | - | - | not explicit | yes |

| *Naming Conflicts* | *renaming* | *implicit renaming* |
|---|---|---|
| *Object/Classes* | Navathe, Batini, Gotthard, Eder | Geller |
| *Types* | Eder | Navathe, Batini, Gotthard, Geller |
| *Attributes* | Navathe, Batini, Gotthard, Eder | Navathe, Batini, Gotthard, Geller |
| *Methods* | Eder | Navathe, Batini, Gotthard, Geller |

| *Scaling Conflicts* | *conversion functions/table* |
|---|---|
| *Attributes* | Navathe, Eder |
| *Methods* | Eder |

| *Structural Conflicts* | *transformation* | *transformation primitive* | *transforming rules* |
|---|---|---|---|
| *Object/Class* | Batini | Gotthard | Geller, Eder |
| *Types* | | Gotthard | Geller, Eder |
| *Attributes* | | Gotthard | Geller, Eder |
| *Methods* | | | (Eder) |

| *Static/Dynamic Conflicts* | Eder |
|---|---|

| *Integration Constraints* | *Navathe* | *Batini* | *Gotthard* | *Geller* | *Eder* |
|---|---|---|---|---|---|
| *class/object identity constraint* | + | +,not explicit | + | + | + |
| *attribute identity constraint* | + | +, not explicit | + | + | + |
| *method identity constraint* | - | - | - | - | + |
| *subclass, selection constraint* | + | - | - | + | + |
| *subtype constraint* | - | - | - | + | + |

| *Redundancy removal* | | | | | |
|---|---|---|---|---|---|
| *redundant classes/objects* | merge to one entity | merge to one entity | integration primitives | structural integration | structural integration |
| *redundant types* | - | - | - | -"- | -"- |
| *redundant attributes* | + | + | integration primitives | -"- | -"- |
| *redundant methods* | - | - | - | -, not explicit | method integration |

| *Schema enrichment* | | | | | |
|---|---|---|---|---|---|
| *type/class/object relationships* | categories | introduce new relation | introduce new relation | + | + |
| *component relationships* | relation merging | schema restruct. | schema restruct. | + | + |
| *method relationships* | - | - | - | - | method integration |