Structural Schema Integration in Heterogeneous Multi-Database Systems using the Dual Model

James Geller, Yehoshua Perl
Inst. for Integrated Systems, CIS and CMS
NJ Inst. of Technology, Newark, NJ 07102

Erich Neuhold
Inst. for Integ. Publ. and Info. Systems
GMD, Darmstadt, FRG (Germany)

ABSTRACT: The integration of views and schemas is an important part of database design and evolution to support complex and multiple applications sharing data. The view and schema integration methodologies proposed and used to date are driven purely by semantic considerations, and allow integration of objects only if that is valid from both semantic and structural view points. We discuss a new integration method called *structural integration* that has the advantage of being able to integrate objects that have structural similarities, even if they differ semantically. This is possible by using the object-oriented Dual Model which allows separate representation of structure and semantics. Structural integration has several advantages, including the identification of shared common structures that is important for physical database design and sharing of data and methods (code reusability).

1 Introduction

As a solution to the important problem of *database integration* the concept of a *federated database* was introduced by Hammer and McLeod [HM79] and Heimbigner and McLeod [HM85]. A federated database system consists of several component databases which can all be accessed through a Federated Database Managemet System (FDBMS). The component systems continue to function as autonomous databases, while at the same time the federation permits controlled access to data from all component databases. A new layer in a heterogeneous environment, called by Kim [K89] a "Heterogeneous Database Integrator," should not result in any changes to existing database systems. Madnick et al. [M89] have refined this notion into *system non-intrusiveness* and *data non-intrusiveness*, meaning that neither system components should be added, nor data in preexisting databases adapted. Sheth and Larson [SL90] present an autonomy oriented taxonomy of federated databases, in which they distinguish between loosely coupled systems (e.g. [RECG89]) and tightly coupled systems (e.g. TBCD87]).

Many approaches to database integration rely on traditional data models, possibly with some extensions, e.g. [EN84,SL88] use extended forms of the ER model. In contrast, it is believed by a number of researchers, that object-oriented databases will aid in the problem of integrating heterogeneous components (e.g. [BM89,K89]). Approaches towards object-oriented integration methodologies have been reported, e.g., in [KDN90,BNPS89,SN88].

When talking about integration in the database context there are two different viable approaches that can be gleaned from the literature. The bottom-up approach, which we refer to as *database integration*, essentially requires that the possibly partial (local) schema of an existing participating database is transformed into the modeling language of the multi-database system. From this transformed schema subsets are created as export schemata to be integrated into one or multiple common (federated) schemata. In the top-down approach, which we refer to as *view integration*, the external schemata of cooperating applications will have to be combined into one or multiple common (federated) schemata to be connected then to newly created databases. As a common name for both approaches we will use the term *schema integration.*

We will show that the object-oriented data models can be improved in their expressive power if the structural and the semantic aspects of a schema can be handled separately. Current schema integration methodologies allow integration of objects only if that is valid from both semantic and structural view points. We discuss a new integration method called structural integration that has the advantage of being able to integrate objects that have structural similarities, even if they differ semantically. We shall use the Dual Model [NGPT90,NPGT89a,NPGT89b] that incorporates mechanisms for a separate treatment of structural and semantic modeling principles. The treatment of the complementary case of semantic similarity and different structure is deferred to a future paper. The structural integration method is not intended to replace integration by generalization, but to introduce an additional tool for integration in cases where current techniques are not applicable.

TH0372-3/91/0000/0200$01.00 © 1991 IEEE

2 Integration by Generalization

As an example environment which has been used widely in the literature we will consider a purchasing department. The activity of a purchasing department is composed of two phases which can be identified with two separate divisions of the department. The first phase is "ordering" which results in a purchase order. The second phase involves receiving the goods and producing a receiving report. To demonstrate the process of schema integration we will present (next to each other) small parts of two separate object-oriented database schemata, one created for the ordering division and one created for the receiving division. Issues that are not absolutely necessary will be omitted.

A class definition is done with a language called VML [FKRST89]. It consists of a class name, immediately following the keyword **class**, and a sequence of properties, given in the order (1) relations, (2) attributes, (3) relationships, and (4) methods. The difference between (1) and (3) is that relations are part of the VML language and as such predefined, while relationships are user-defined. Space limitations prohibit a complete explanation of the VML language. We rely on the intuitions of the reader. We also need to limit ourselves to the crucial parts of the example code. Specifically, we omit the classes purch_orders, receiv_reports, and supplier.

ORDERING DB	RECEIVING DB
class purch_order	**class** receiv_report
member: purch_orders	**member:** receiv_reports
attributes:	**attributes:**
OrderNumber: INTEGER	OrderNumber: INTEGER
Amount: INTEGER	Amount: INTEGER
Unit: STRING	Unit: STRING
Price: DOLLARTYPE	Price: DOLLARTYPE
OrderDate: DATETP	ReceivDate: DATETP
relationships:	**relationships:**
Item: catlg_item	Item: receiv_item
Refer: receiv_report	Refer: purch_order
class catlg_item	**class** receiv_item
attributes:	**attributes:**
CatlgNumber: INTEGER	CatlgNumber: INTEGER
Name: STRING	Name: STRING
relationships:	**relationships:**
OfferedBy: supplier	OfferedBy: supplier
Orders: purch_orders	Reports: receive_reports

In order to clearly demonstrate the advantages of structural integration we need to compare it to the currently popular strategy of integration by generalization [LR88,AM87]. With the usual generalization mechanisms the common properties of two or more classes (from different applications or databases) are identified and collected into a superclass where the original classes become subclasses and contain only those properties that are specific to them. For database integration the subclasses would come from the export schema to be mapped into the local database schema. For view integration the subclasses would come from the user views. In both cases the generalized superclass would be a member of the federated common schema.

In the above example we can identify two sets of very similar class definitions, i.e., (purch_order, receiv_report) and (catlg_item, receiv_item). In the first case we could use generalization to produce a superclass such as "purchasing forms," but we argue that this class is not a proper semantic generalization of the two classes purch_order and receiv_report. Each of these two classes expresses very different information and is handled (via methods not shown here) differently. The generalization is actually only a structural one, but we cannot find that fact without an elaborate analysis of the semantics of the two classes at the time when we want to utilize the unified schema.

In contrast, the two classes catlg_item and receive_item not only describe objects of similar structure but also of similar semantics. Here they actually denote the same objects at different phases of the purchasing process. Generalization allows us to define a class hierarchy of the following form and to save in the amount of specification necessary.

class item
attributes:
CatlgNumber: INTEGER
Name: STRING
relationships:
OfferedBy: supplier

class catlg_item	**class** receiv_item
roleof: item	**roleof:** item
relationships:	**relationships:**
Orders: purch_orders	Reports: receiv_reports

The class hierarchy here abstracts both the semantics of the classes catlg_item and receive_item, as well as their structural descriptions. In the integrated database there may now exist instances of items that are neither ordered nor received but just e.g. members of some supplier's catalog.

With generalization a single subclass hierarchy has to be used for two purposes at the same time: (1) to factorize common structure and behavior of classes and (2) to express additional semantic relationships between classes. This leads to problems when attempting to integrate structurally similar but semantically dissimilar classes. But it also leads to a situation that two classes modeling semantically related objects can only be dealt with, if the objects in question are structurally similar as well. The use of a single hierarchy for two conceptually distinct connections among specifications has resulted in inadequate conceptual models. Therefore, it should be advantageous to separate those two parts of the specification, and we have done so in the Dual Model.

3 Structural Integration Using the Dual Model

We will now present a few of the highlights of the Dual Model to give the reader a better understanding of its usefulness. In order to express that all instances of a class have a common structure and behavior we consider them to be of the same abstract data type. This type is called the *object type* of that class. Hence, we associate with each object class an object type. Different classes may have the same object type. A *type-to-class mapping* provides the necessary interface between object types and corresponding classes. An object type is determined by a list of properties. There are four kinds of properties, and they correspond to the already introduced class properties. Attributes contain values of a given type. Relationships contain references to other object types. Methods are code segments to be used on the instances of the object type. Relations contain references to other object types and are system defined. The object types can be organized in a type hierarchy utilizing structural inheritance for properties [NPGT89b].

The classes themselves reflect the object instances and the semantic constraints and semantic relationships between classes which can be formulated independently of the object types. The classes are organized in a hierarchy utilizing semantic inheritance [NGPT89b]. It is important to point out that the terms *structure* and *semantics* are used differently than in other data models. A more exact definition of our use can be found in [GPCS91].

In the following examples the left column represents the object types of the Dual Model, and the right column the corresponding object class(es). In the left column definitions are omitted, if there exists only one class for a given type. The names of object types are printed with capital letters, the names of object classes are printed with small letters, properties have only the first letter capitalized.

Structural schema integration is based on identifying structural similarities between the two object types which represent the structural elements of the two corresponding classes while disregarding the semantic aspects of the two classes. In case these two object types are equal we have a case of structural schema integration since we found a common structural object type for these two classes. Such integration will be possible even where schema integration by generalization is not possible due to semantic differences between the classes. For reasons of space a full formal definition of structural integration will be given in [GPCS91,GNP91].

Returning to the example from Section 2 it turns out that structural integration can be performed without combining purch_order and receiv_report into a superclass. Looking at the structure of purch_order and receiv_report we realize that they are identical except for four differences. If we resolve these differences we can assign the same object time, PURCH_FORM, to purch_order and receiv_report. Both classes have *member* relations to the corresponding set classes purch_orders and receiv_reports. They have the relationship Item to catlg_item and receive_item, respectively. They have the relationship Refer (Reference) with which they refer to one another and they are different in the attribute Date.

The difference between purch_orders and receiv_reports is settled if we define an object type PURCH_FORMS for both purch_orders and receiv_reports, which would have as instances sets of objects from purch_order and receiv_report respectively. The relationship Item can be made to refer to an object type ITEM, *iff* catlg_item and receiv_item can be integrated by this object type ITEM. The third difference is settled automatically once we have an object type PURCH_FORM, because in PURCH_FORM the relationship Refer will point to PURCH_FORM itself. The actual classes purch_order and receiv_report utilizing PURCH_FORM will be specified in the semantic description in the right column of our specification. This semantic description of each class specifies the actual classes of the object types ITEM and PURCH_FORM referred to by the relationships Item and Refer, respectively. *Date* is transformed by the previously mentioned type to class mapping to OrderDate for purch_order and to ReceivDate for receiv_report.

objecttype PURCH_FORM	**class** purch_order
member: PURCH_FORMS	**objtp:** PURCH_FORM
attributes:	**member:** purch_orders
OrderNumber: INTEGER	**attributes:**
Amount: INTEGER	OrderDate: DATETP
Unit: STRING	**relationships:**
Price: DOLLARTYPE	Item: catlg_item
Date: DATETP	Refer: receiv_report
relationships:	**class** receiv_report
Item: ITEM	**objtp:** PURCH_FORM
Refer: PURCH_FORM	**member:** receiv_reports
	attributes:
	ReceivDate: DATETP
	relationships:
	Item: receiv_item
	Refer: purch_order

The object type PURCH_FORM expresses the structural integration of the two classes purch_order and receiv_report, thus achieving the desired savings in the specification. Now we show that both, catlg_item and receiv_item can actually have a common object type ITEM, thus enabling the object type PURCH_FORM as described above. The relationship Forms is transformed by the type to class mapping to "orders" and "reports."

objecttype ITEM	**class** catlg_item
attributes:	**objtp:** ITEM
CatlgNumber: INTEGER	**relationships:**
Name: STRING	Orders: purch_orders
relationships:	**class** receiv_item
OfferedBy: SUPPLIER	**objtp:** ITEM
Forms: PURCH_FORMS	**relationships:**
	Reports: receiv_reports

For both pairs of classes, (purch_order, receiv_report) and (catlg_item, receiv_item) we had a case of full structural correspondence, formally defined in [GPCS91,GNP91]. Also defined there is partial structural correspondence which permits structural integration of classes with minor structural differences by one single object type.

Notice that our example has shown that it is sometimes not possible to perform structural integration for a single pair of classes, but that two interrelated subsets of classes need to be analyzed to allow for structural integration.

In our example we have shown structural integration as an *ad hoc* method. In [GNP91] we will elaborate on a methodology for this process which tends to become quite complicated for large databases. This methodology is based on using a labeled graph model of the object-oriented database schema, where classes and object types are represented as nodes, and relations and relationships as edges, and structural integration makes use of graph isomorphisms.

In this paper we have introduced a new technique called structural integration. We have demonstrated that this technique is capable of integrating classes with similar structure and different semantics, for which integration by generalization is not applicable. Thus, structural integration can serve as an additional integration tool besides generalization.

REFERENCES:

[AM87] Andrews, T. and Morris, C., "Combining Language and Database Advances in an Object-Oriented Development Environment", Proceedings of the OOPSLA Conference, 1987.

[BM89] Brodie, M. L., and Manola, F., Database Management: A Survey. Readings in Artificial Intelligence and Databases, edited by J. Mylopoulos and M. L. Brodie, Morgan Kaufmann Publishers, San Mateo California, 1989.

[BNPS89] Bertino, E., Negri, M., Pelagatti, G., Sbattella, L., An Object-Oriented Approach to the Interconnection of Heterogeneous Databases. 1989 Workshop on Heterogeneous Databases, Sponsored by NSF, Northwestern University and IEEE-CS, 1989.

[EN84] Elmasri, R., and Navathe, S., "Object Integration in Logical Database Design", Proceedings of the 1st Intl. Conference on Data Engineering, 1984.

[FKRST89] Fischer D., Klas, W., Rostek, L., Schiel, U., Turau, V., "VML - The VODAK Data Modelling Language", GMD-IPSI, Technical Report, Dec. 1989.

[GPCS91] Geller, J., Perl, Y., Cannata, P., and Sheth, A. "Structural Integration using the Object-Oriented Dual Model", Submitted for Publication.

[GNP91] Geller, J., Neuhold, E.J., and Perl, Y. "Full and Partial Correspondence in Structural Schema Integration", In preparation.

[HM79] Hammer M., and McLeod D., "On Database Management
System Architecture", Tech Report, MIT/LCS/TM-141, Massachusetts Institute of Technology, Cambridge, MA, 1979.

[HM85] Heimbinger, D., and McLeod, D., A Federated Architecture for Information Management, ACM Trans. on Office Information Systems, 3,3, 253-278, 1985.

[K89] Kim, W., Research Direction for Integrating Heterogeneous Databases. 1989 Workshop on Heterogeneous Databases, Sponsored by NSF, Northwestern University and IEEE-CS.

[KDN90] Kaul, M., Drostern, K., and Neuhold, E., "ViewSystem: Integrating Heterogeneous Information Bases by Object Oriented Views", In Proceedings of the 6th Intl. Conf. on Data Engineering, (Los Angeles, CA), 1990.

[LR88] Lecluse, C. and Richard, P., "Modeling Inheritance and Genericity in Object-Oriented Databases", LNCS #326, ICOT 1988, p. 223-237.

[M89] Madnick, *et al.* CISL: Composing Answers from Disparate Information Systems. 1989 Workshop on Heterogeneous Databases, Sponsored by NSF, Northwestern University and IEEE-CS.

[NGPT90] Neuhold, E. J., Geller, J., Perl, Y., Turau, V.. "A Theoretical Underlying Dual Model for Knowledge-Based Systems", The First International Conference on Systems Integration, Morristown, NJ, 96-103, 1990.

[NPGT89a] Neuhold, E. J., Perl, Y., Geller, J., Turau, V., "Separating Structural and Semantic Elements in Object-Oriented Knowledge Bases", Advanced Database System Symposium, Kyoto, Japan, 1989, 67-74.

[NPGT89b] E. Neuhold, Y. Perl, J. Geller, V. Turau, "The Dual Model for Object Oriented Knowledge Bases", New Jersey Institute of Technology, Tech Report CIS-89-23, submitted for publication.

[RECG89] Rusinkiewicz, M., Elmasri, R., Czejdo, B., Georakkopoulous, D., Karabatis, G., Jamoussi, A., Loa, L., and Li, Y., "OMNIBASE: Design and Implementation of a Multidatabase System." In Proceedings of the 1st Annual Symposium in Parallel and Distributed Processing, Dallas, Texas, 162-169, 1989.

[SL88] Sheth, A. P., and Larson, J. A., "A Tool for Integrating Conceptual Schemas and User Views," Proceedings of the Fourth Intl. Conf. on Data Engineering, Los Angeles, CA, February 1988.

[SL90] Sheth, A. P., and Larson, J. A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, 1990, 183-236.

[SN88] Schrefl, M. and Neuhold, E. J., "A Knowledge-based approach to overcome structural differences in object-oriented database integration", Proceedings of the IFIP Working Conference on The Role of Artificial Intelligence in Database and Information Systems, Canton, China, North Holland Publ. Co., 1988.

[TBCD87] Templeton, M., Brill, D., Chen, A., Dao, S., Lund, E., McGregor, R. and Ward, P., "Mermaid: A Front-end to Distributed Heterogeneous Databases, Proceedings of the IEEE, 695-708, April 1987.