Research into Networks and Distributed Applications
R. Speth (Editor)
Elsevier Science Publishers B.V. (North-Holland)

GLOBAL VIEW DEFINITION AND MULTIDATABASE LANGUAGES - TWO APPROACHES TO DATABASE INTEGRATION+

Peter FANKHAUSER*
Witold LITWIN‡
Erich J. NEUHOLD†
Michael SCHREFL†

*University of Vienna
†GMD, Darmstadt
‡INRIA, Rocquencourt

The paper deals with new aspects of two different approaches to achieve integrated access to autonomous heterogeneous databases. One is to assist the user in the definition of his personal view over different databases. The other one is to provide new features to multidatabase manipulation languages for manipulation of distinct and mutually not integrated databases.Both provide the capability to use multiple databases without the need to predefine a complete global schema. The two approaches are related to each other.

1. INTRODUCTION

Integrating access to preexisting *data-model-heterogeneous* DBs can be split into two tasks. The first one is to achieve data model homogeneity. Therefore it is required (i) to map the local conceptual schemata (or parts of them) to *export schemata* in a canonical data model, (ii) to specify the translation of queries against the export schemata into according queries against the local schemata and (iii) to define the mapping of the responses in the format of the local schemata into the format of the export schemata. The data model homogeneous export schemata may still be *semantically* heterogeneous. They may model similar or even the same type of data with differences in naming, scaling, in the level of detail and in the way of data abstraction. Thus the second task is to overcome the *semantic heterogeneity*.

In this paper we concentrate on the latter task. In order to overcome the semantic heterogeneity either semantically homogeneous *global views* may be defined [3, 9, 10, 18] or a *multidatabase manipulation language* [6, 7] capable of manipulating jointly data in different databases may be supplied. The first possibility is useful for repetitive manipulations. The second one provides the advantage to be flexible with respect to ad hoc queries.

In the following both approaches are introduced. In the second section an outline of useful operators for defining integrated global views is given and some of the implied problems are discussed. In the third section necessary extensions to database manipulation languages are discussed and they are illustrated by some examples. The paper is concluded by a relating the two approaches to each other.

+ This work was partially sponsered by the Commission of European Communities under Cost 11ter project IHIS and the Austrian science foundation (Fonds zur Foerderung der wissenschaftlichen Forschung) under grant P 5976 P.

2. PERSONALIZED GLOBAL VIEWS

The framework for the research described in this section is a project on knowledge oriented distributed information management (KODIM). KODIM is intended to provide information management facilities to support homogeneous access to private and public databases by establishing global views. A single global schema that integrates completely the local schemata of a large number of databases is toilsome if not impossible to establish and will hardly meet any users intention. Contrary, a personalized global view describes a multidatabase view on those parts of some databases which are of interest to a particular person.

2.1. The scenario

In a *preintegration phase* the export schemata which we assume to exist already for the purpose of this paper are extended by some meta database information. The local database administrators add for example domains for attributes, constraints, semantic relationships between classes, synonym names and access rights to make the local databases self describing.

Then the various database administrators communicate with each other to define partially the initial global view. The initial global view will at least consist of a set of primitive global datatypes and the appropriate transformation rules between global and local datatypes.

An individual user or user group may define its view upon a general initial global view and several export schemata. A personalized view should not be regarded as a static schema. It rather dynamically evolves when a user accesses a multidatabase system and thereby increases his knowledge about the databases' contents. A user may have some query in mind but may not yet know how to pose it correctly. He may not be aware of the names used for the classes and attributes of some export schema. He might even not know which databases to use. In this case a knowledge navigator will assist the user in selecting appropriate databases and identifying relevant classes and attributes. If these classes reside in different export schemata the user is assisted to integrate them into a semantically homogeneous representation in his personalized view. The knowledge navigator uses metaknowledge like a taxonomy of terms (thesaurus) for which each database holds information, a reduced description of the several local conceptual scemata and other relevant information.

It has been proposed to use the relational model as canonical model as it will be supported by many commercial database systems. It is simple, easily understood and subject to standardization. However, it was basically developed for tabular data, and has been found not to be that adequate to model other types of data frequently used. For instance in the field of document databases the domains of text search fields are solely defined by the retrieve operators which may be applied on them. These retrieve operators can have similar function (e.g. retrieving from full text) in several databases but usually differ in their syntax and their semantics with respect to proximity operators, thesauri, etc.. Mapping the contents of documents and the according queries into the relational model is bordersome and inefficient.

We propose [4, 11, 12, 13] to use an *object oriented* data model as canonical model. Object oriented data models [1, 2, 17] encapsulate data structures, termed *object classes*, with their (specialized) operations, termed *methods*. Operations are executed on an instance of a class by sending the object class a message which identifies a method selector and possible argument values. A message will be denoted as:

['object-identifier' *method-selector:* {argument-values}].

The set of messages to which an object can respond is called *interface* of the object and the set of possible reactions *behaviour*. Global views - global object classes - derive their behaviour fully from (local) object classes of the export schemata or other already established global views.

In the next subsections several concepts to establish global views are introduced.

2.2. Object class integration by generalization

In many cases global views constitute a generalization of exported object classes with some common behaviour. Consider for example the selections of the order handling databases of two companies in Figures 1 and 2. The class PART of company A has the attributes "PartNo", "ManufacturedFirst" and "Weight", and the method "Price" which calculates the price of a product in POUND according to the practice of company A. Company B has modelled a comparable entity PRODUCT, but has used different names for methods and attributes with comparable functionality, e.g. it uses "Print" for displaying a PRODUCT whereas company A uses "Show" for displaying a PART. Furthermore the prices of parts and products are represented in different currencies.

class PART
    attributes:
        PartNo: STRING
        ManufacturedFirst: YEAR
        Weight: KILO
    methods:
        Price: POUND
        Show:

class CUSTOMER
    attributes:
        Name: STRING
        Address: STRING
        Credit: POUND
        ...

Figure 1: Selection from the order handling database of Company A

class PRODUCT
    attributes:
        ProductNo: STRING
        ManufacturedFirst: YEAR
    methods:
        Price: DM
        Print:

class CUSTOMER
    attributes:
        Name: STRING
        Address: STRING
        Credit: DM
        ...

Figure 2: Selection from the order handling database of Company B

To operate the databases after a merger of the two companies jointly the differences in attributes and methods will become burdensome. A view offering a single set of data manipulation functions (messages) should be available. This is achieved by the definition of a common generalization class for two or more related object classes. The most flexible way to specify the global behaviour of the instances of the related object classes is to define explicitly all methods at the generalization classes and combine their results properly. However if there exists a large number of methods this task becomes burdensome. Therefore we use *upward inheritance* [15] by searching for methods not found at a (global) generalization class at its subclasses. To combine methods inherited upward from more than one subclass appropriately by default rules we distinguish several generalization types:

CATEGORY GENERALIZATION
Classes that model different real world objects which share some common properties (methods) but cannot be identified by any of these constitute categories. Consider for example classes PART and PRODUCT from Figure 2 which are category generalized to the global class PRODUCT in Figure 3.

```
class PRODUCT
    metaclass:          CATEGORY_GENERALIZATION_CLASSES
    generalization of:  A.PART, B.PRODUCT
    attributes:
        OrderNo
            corresponding:
                A.PART.PartNo
                B.PRODUCT.ProductNo
    methods:
        Display
            corresponding:
                A.PART.Show
                B.PRODUCT.Print
        Price:          DOLLAR
```

Figure 3: The category generalized class PRODUCT

The global class PRODUCT is made an instance of the metaclass CATEGORY_GENERALIZATION_CLASS. Only the combination of synonymeously named and differently scaled methods and attributes has to be specified explicitly. The rest is inherited upward and combined according to default rules specified with the metaclass. The method specification "Display" says to display products of company A by the message "Show" and products of company B by the message "Print", and the method specification "Price" states that the price of a product is to be given uniformly in DOLLAR.

ROLE GENERALIZATION
Classes that model the same real world entities in a different situation or context can be role generalized.

```
class CUSTOMER
    metaclass:                  ROLE_GENERALIZATION_CLASSES
    generalization of:          A.CUSTOMER, B.CUSTOMER
    object correspondence rules:
        A.CUSTOMER.Name = B.CUSTOMER.Name and
        A.CUSTOMER.Address = B.CUSTOMER.Address
```

attributes:
Credit: DOLLAR

Figure 4: The role generalized class CUSTOMER

In Figure 4 the class CUSTOMER is defined as role generalization of the classes A.CUSTOMER and B.CUSTOMER. As both classes have been instantiated independently they will represent the same real world person by different object identifiers and several object identifiers exist for one and the same instance of the generalization class. In order to be able to relate equivalent object identifiers, *object corresponding rules*, which are predicates that evaluate to true for corresponding instances of the local classes, are associated with role generalization classes. In cases where the correspondence cannot be expressed by a generic predicate at class level it has to be instantiated expicitly for objects found to correspond with each other.

IDENTITY GENERALIZATION
Object classes that model the same set of real world objects at the same point of time can be identity generalized to global views without any change. This special case will hardly occur. Nevertheless the operation makes sense because of the restriction not to violate the autonomy of the export schemata. If a user wants to include a local object class as it is into the global view in order to further semantically relate it to other global views he can identity generalize it alone and feel free to modify it further.

HISTORY GENERALIZATION
Classes that model the same real world objects at different points (intervals) of time can be history generalized (e.g. *employees* who are modeled as former *students* in a university's database and are now modeled in their company's database).

COUNTERPART GENERALIZATION
Object classes that model disjunctive sets of objects which share some common properties but represent alternative situations in the real world can be counterpart generalized (e.g. *air-connections* and *train-connections* correspond with each other according to their departure- and arrival-city and can therefore be generalized to *travel-connections*)

When an object is retrieved by a query against a global view it should continue to follow the global behaviour that query. In order to distinguish between the local and global behaviour of an object the concept of *object coloring* is used. An object follows the behaviour of a certain object class when it is tinged by a color unique for this object class. The color of an object identifier is denoted by a superscript to the identifier, e.g. 'Meier-$A^{CUSTOMER}$'. The special message "*as:* object-class-name" sent to an object changes its color accordingly. The corresponding method "as" is defined at and inherited from the metaclass of the generalization class constituting the global view.

Additionaly when inherited attributes are scaled in a different way a unique scale has to be chosen for the global view. In order to switch between the local and global representation of the corresponding values (objects) the concept of *object transformation* is used. It is assumed that every instance of a local data type class will respond to the message

['data-value' *in:* data-type-class-name]

by first coloring the data-value by the data-type-class-name and sending itself the mesage "TransformFrom", i.e.:

[['data-value' *as:* data-type-class-name] *TransformFrom*].

The methods "TransformFrom" and "TransformTo" are inherited from the system defined meta class DATA_TYPE_CONVERSION_CLASS and use the actual transformation methods specified with the individual global data type class. Note, that an asymetric approach is necessary because modification of

local classes has to be avoided and the appropriate transformation methods have to be specified with the global class (Figure 5).

class DOLLAR
metaclass: DATA_TYPE_CONVERSION_CLASS
generalization of: POUND, DM
transformation methods:
FromPound
FromDM
ToPound
ToDM

Figure 5: The global view DOLLAR

For example the message ['5DM' in: DOLLAR] will be transformed into [['5DM' *as:* DOLLAR] *TransformFrom*]. The colored object '5DM^{DOLLAR}' follows the behaviour of the global class DOLLAR. Therefore it can respond to the "TransformFrom" message, which is inherited by DOLLAR from DATA_TYPE_CONVERSION_CLASS, and use the transformation method "FromDM" to convert '5DM^{DOLLAR}' into '3$'

Consider now that all customers named "Meier" are retrieved and assigned to a variable by the expression:

meier := [CUSTOMER *where:* Name = "Meier"].

Note that it is assumed that every class can respond to the special message *where* in order to select objects by a qualification given as parameter. In order to access the concrete instances of the local object classes the message is transformed into

[A.CUSTOMER *where:* Name = "Meier"] *as:* CUSTOMER] *union*
[[B.CUSTOMER *where:* Name = "Meier"] *as:* CUSTOMER]

The interpretation of *union* specific for role-generalized objects uses the given object correspondence rules to relate equivalent objects and chooses deliberatly one of the corresponding object identifiers to represent the generalized object. The *as*-message is used to color the resulting objects according to the global context of the query. Now the credit of the retrieved customers can be obtained by the message

[meier *Credit*]

When a certain member of the set in meiers, let it be 'Meier-A^{CUSTOMER}', receives the message *Credit* it will retrieve its color and realize that it has to follow the behaviour of the role generalized class CUSTOMER. There it will find that if it is instance of A.CUSTOMER and B.CUSTOMER, which is the case in our example, it has to union multiply inherited credits of both related objects. Thus the message is transformed into

[[['Meier-A^{CUSTOMER}' *as:* A.CUSTOMER] Credit] *in:* DOLLAR] *union*
['Meier-A^{CUSTOMER}' *as:* B.CUSTOMER] Credit] *in:* DOLLAR]]

The object transformation from POUND in DOLLAR and from DM in DOLLAR has to take place because the local object classes use different currencies. When *union*-message has been implemented as summation (operator overloading) for the data type class DOLLAR the total credit given by both companies to Mr. Meier is returned.

2.3. Qualified inheritance

With inheritance along the well known *is-a* relationship it is assumed that *all* methods and attributes of a superclass are propagated to its subclasses unless it is overridden at a subclass explicitly. With the reverse relationship *generalization-of* we assume as well that all attributes and methods are inherited upward. This concept of *full* inheritance can be refined by associating to other semantic relationships (e.g. *part-of*) a message "Propagates" which is implemented as a logic expression serving as selective qualification on methods and attributes to be inherited over that relation (Figure 6). That logic expression is evaluated by the message handling mechanism when an attribute or method not found at an object is searched for and found at some semantically related object. If it evaluates to true the considered attribute or method is inherited. In the implementation of "Propagates" the methods and attributes can either be explicitly enumerated (connected by "or") by name or selected by some predicate. For example with the PART_OF relationship each candidate <attribute> for inheritance is tested for belonging to the metaclass PART_OF_INHERITABLE_ATTRIBUTES. If an attribute is not associated to an appropriate metaclass or a semantic relationship has got no method "Propagates" no inheritance is assumed. The concept corresponds to the notion of include schemata in CRL™ (Carnegie Representation Language) [5] in KnowledgeCraft™.

class: GENERALIZATION_OF
  metaclass: RELATION_CLASSES
  methods:
    Propagates: BOOL
      implementation: TRUE

class: PART_OF
  metaclass: RELATION_CLASSES
  methods:
    Propagates: BOOL
      implementation:
        [[<attribute> *metaclass*] = PART_OF_INHERITABLE_ATTRIBUTES]

Figure 6: semantic relationships with qualified inheritance

The object classes of the export schemata can be semantically enriched in the preintegration phase by associating their methods to appropriate metaclasses. For instance in Figure 7 the attribute "Owner" of the local object class CAR has been enriched appropriately.

class: CAR
  attributes:
    Owner: PERSON
      metaclass: PART_OF_INHERITABLE_ATTRIBUTES
    MotorId: STRING
    Color: STRING
    ...

Figure 7: semantically enriched object class CAR in database of company A.

The user then has at his disposal a number of predefined semantic relationships to express his intentions for a global view. For instance if an already established global view MOTOR is related to the

above object class CAR as in Figure 8 it additionally inherits all part-of-inheritable attributes.

class: MOTOR
...
part_of: A.CAR
attributes:
Power: KJOULE
Id: STRING
...

Figure 8: use of semantic relationship PART_OF

When some motor is sent the message *Owner* which it cannot handle directly the message is forwarded along the semantic relation PART_OF to an associated car where an appropriate attribute is found. The inheritance specification of PART_OF evaluates to true therefore in order to respond to the original request the according value is propagated to the motor. Note that as with role generalized objects the part-of relation has to be explicitly instantiated if it is not possible to relate objects by a corresponding rule over user defined keys at object class level.

2.4. System assisted derivation of methods

If databases have been designed autonomously the same universe may be modelled in different ways. Differences in logical data structures result for example when some fact is modelled once as attribute and once as object. In these cases the comprehensive behaviour derivation by inheritance is too inaccurate. In order to overcome structural differences and to derive the behaviour of global views very specifically the system assists the user to implement the according methods.

For this purpose the concept of *message forwarding* [14] is used. When a message cannot be handled by an object directly - due to structural differences or not sufficiently established semantic relationships - it is forwarded as inquiry message at class level along related objects that can handle the message. Thereby the path(s) are recorded which serve - if the search has been successful - as basis for a (number of) plan(s) to implement the desired method. A set of message forwarding rules guides the search and a set of plan combination rules assists the user in choosing appropriate plans and combining them. The process to determine a message forwarding plan is related to the problem to determine implicit joins in queries against a (multi)relational database [8].

Once a plan is defined it serves as method until it becomes obsolete because of subsequent changes of some export schemata. Then the plan is presented for modification or a completely new plan can be derived as described above.

The concept of object coloring from section 2.2. is generalized to the concept of *context coloring* in order to remember with an object the environment in which it has been retrieved. The object that is returned to a forwarded message is colored by the objects which identify the context through which the message has been forwarded. Thereby it reflects the environment in which it was originally retrieved in its subsequent behaviour. Consider the following scenario. A product some customer has ordered is retrieved by a message to the customer. The same product may have been ordered by a number of other customers as well. Nevertheless in the context of the actual query the product with respect only to the considered customer is of interest; the retrieved product depends on that customer. If a subsequent message sent to the retrieved product asks for the order date the product takes into account the context of the previous message and will respond with the date on which the considered customer, and

not another one, has ordered the product.

Note that the above concept results in a weaker version of inheritance. Whereas the system exploits the combination rules given by generalization metaclasses (2.2.) and the inheritance qualification associated to semantic relationships (2.3.) automatically, both combining and cutting out possible plans at has to be partially confirmed by the user. Upward inheritance, qualified inheritance and system assistance to implement methods explicitly together form a rich set of tools to establish and dynamically refine global views.

3. MULTIDATABASE MANIPULATION LANGUAGES

3.1. New needs

Database languages were intended for the definition and manipulation of data within a database. A multidatabase language has the same goals, but for a collection of databases. This extension requires new capabilities responding to the following new needs:

(1) - in general, if a manipulation is expressible in a data model for data are in a database, it should be expressible as well for these data spread in any way over different databases. For the relational model, it should be in particular the case of joins.

(2) - the user should be able to indicate databases to be used. The language should provide therefore the notion of the database name. The database name may in particular be needed to solve name conflicts between data in different databases.

(3) - while data in a database are assumed integrated, they cannot be in general assumed integrated for different databases. The databases are indeed autonomous and the primary goal of each database are the local needs. A multidatabase query may then need to specify a common manipulation of data that unlike in a database are redundant or differ with respect to names, value types or structure. The language has to allow the user to formulate such manipulations in a non procedural way despite these discrepancies.

(4) - for the similar reasons, the user should be able to define data in a cooperative way. This involves the definition of data in multiple schemas in a single statement or the definition of access right imported from another database etc.

(5) - the user should be able to move data between databases. These data may need to be gathered from several databases and included into several others as well. The transfer may include data conversion.

(6) - the user should be able to define (i) various types of dependencies among selected databases and (ii) import schemas (global views).

The overall purpose of these capabilities is to make simple the usage of multiple databases without any global (import) schema. It is felt that to make the requirement for such a schema mandatory prior to any multidatabase manipulation of a collection of databases, would greatly restrict the user flexibility. In contrast, the integrated presentation may be useful for repetitive and basically known in advance manipulations. This observation comes out from experiments with the prototype multidatabase system MRDSM and is shared by database systems manufacturers. Most of new major systems provide indeed

the basic multidatabase capabilities (1) and (2) above. This is especially the case of Sybase, of Oracle V5 and of Empress V2, though the corresponding syntax differs. The Oracle provides in particular extensive capabilities for interdatabase queries (5). In contrast, Sybase allows to define interdatabase manipulation dependencies in the form of interdatabase triggers.

3.2. Design problems

The new capabilities require solutions to two aspects of a (multi)database language design:

At the user interface level:
New concepts have to be introduced for the formulation of various types of statements. For instance, in addition to the concept of the database name, it is useful to admit the possibility of multiple identification of data types to deal with redundant data sharing names. It is also useful to allow the user to define new data names in statements for a common manipulation of data that have different actual names. Then, the user should dispose of powerful functions for instance conversion or homogenization of value types, for unit specification etc.

At the implementation level
New capabilities require technical solutions at the implementation level that were not considered for database systems or insufficiently developed as not required strongly. For instance, to bind data name used by the user to the actual names, some analyses should be done, including sometimes the access to an external thesaurus. On the other hand, a flexible value type conversion for both retrievals and updates may require the access to systems for symbolic calculus like Macsyma and/or to equation solvers. It also requires efficient methods for unit conversion that do not exist yet in database systems, since the actual units of data to be gathered, compared or updated by a statement may differ from one database to another. These and similar problems were supposed not to exist in a database, as the task of the database administrator was precisely to remove such discrepancies.

3.3. Examples

To illustrate both types of problems consider the following queries.

1. The user wishes to know the wages of Policemen and of Firemen which are recorded in two tables named accordingly in distinct databases also named accordingly. He knows that the corresponding columns are called *wage* for Policemen and *wages* for Firemen. To do it in a single query, one way is to allow the user:

- to open both databases simultaneously,
- to name both tables through a single name, let it be X. If the user prefers meaningful names, he would choose Employees for instance.
- to use generic characters like '%' for any sequence of characters in data names.

If SQL is extended by these features, to the language we termed MSQL, the query may be simply as follows:

```
USE Policemen Firemen          /*Choice of the databases
LET X BE Policemen Firemen     /*Common name for the tables
SELECT wage%                   /*Double selection
```

The result of the query would be two tables defined by the classical SQL statements obtained through the substitution of actual names to X and executed in any order:

```
/* Database Policemen          /* Database Firemen

SELECT wage                    SELECT wages
FROM Policemen                 FROM Firemen
```

2. Consider now a similar query to ultimately all databases where there are wages of the French state employees, collectively named Employees multidatabase. Assume further that the local autonomy allows each database to call its employee tables as the administrators prefer: Teachers, Workers, Physicians,... . The wages may also be called salary, payment,... The above simple formalism at the language would not suffice anymore. Rather the user query should be simply:

```
USE Employees
SELECT wage
FROM Employees
```

To find the corresponding actual names in the databases, the systems should dispose in contrast at the implementation level of the service of a (sub)system exploring a general purpose thesaurus. The query to each database would be rephrased accordingly. The Thesaurus could be a database by itself.

Consider now that the Firemen wages are indicated with 10 % Social Security payment, while Policemen wages are net. A user wishes to increase by 5 % the wages whose net value is under 5000 F. One way to formulate the corresponding query is to dynamically define for the Firemen a column with net wages and then to update all net wages. In MSQL this may be done as follows:

```
USE Policemen Firemen          /*Choice of the databases
LET X BE Policemen Firemen     /*Common name for the tables
D-COLUMN wage = 0.9 * wages    /*Declaration of the dynamic colum


UPDATE X
SET wage = wage * 1.05         /* Update of the net wages
WHERE wage < 5 000
```

The capability of defining the dynamic columns is new with respect to the SQL user interface. The statement would update both the actual columns Policemen.wage and Firemen.wages. The latter update would require also the capability of symbolic calculus at the implementation level. The definition of wage has indeed to be inverted for the update to the form wages = wage / 0.9.

4. Finally, consider a user wishing to know in FF the wages of Policemen from databases in various European countries, recorded using the local currencies. The formalism of dynamic columns would not suffice anymore. A natural solution is to allow the system to have at the implementation level the access to a (sub)system in charge of currencies conversions. All together, the system would convert by itself the actual currency units to that chosen by the user.

4. CONCLUSION

Multidatabase languages should allow users to formulate multidatabase manipulations in a flexible way and possibly in a single statement. This, no matter what are the actual names or value types that may be heterogeneous from one database to another.

When the semantic differences between data structures of local databases reach a certain degree of complexity the approach of global view definition will be more benificial. The user is mainly interested in getting responses to a query. Thus the process of view definition has to be well integrated into the process of querying at the user interface level. For an enduser view definition should only occur when necessary to respond to a query. Additionaly he should not be burdened with all the details of defining a view.Therefore semantic knowledge is needed. Part of it can be specified in a preintegration phase without anticipating the details of some user's personal views. By establishing semantic relationships that propagate a broad range of behaviour by inheritance the user can specify a view by possibly a single statement. Only those parts of a view's behaviour which do not meet the users intention have to be further specified. The evolution of a personalized global view can furthermore be assisted by means to navigate through the global and local schemata along semantic relationships in order to derive more specific complex methods.

As also stated in [16] semantic knowledge is as well benificial for resolving ambuigities in translating view updates in updates against the underlying classes. A future issue therefore is to extend the concept of semantic relationships and generalization types to addionally capture knowledge about update translation.

In the multidatabase language approach a view may be defined as stored (multidatabase) query, which is evaluated by query modification. In the worst case this requires explicitly defining all dynamic columns and join qualifications (the behaviour) of the view and makes its modification and evolution a complex editing task. These difficulties partially can be solved by accessing auxiliary thesauri and unit conversion systems. Additionally some of the introduced concepts for semantic integration may also apply to the (multi)relational approach by implementing powerful operators which generate a completely defined view according to the assumed semantic relationship between the underlying relations by deriving dynamic columns and inserting appropriate (outer) join predicates by default. Thereby the introduced concepts for view definition would form another layer on the layer which a multidatabase language constitutes on conventional database manipulation languages.

So the two approaches that originally followed a different integration philosophy meet each other at the point where view definition and querying dynamically dissolve into each other.

Acknowledgements

We wish to thank the participants of the IHIS cooperation group for the fruitful discussions and M. Kaul from GMD Darmstadt for carefully reading the paper and helpfully commenting it.

References

1

Serge Abiteboul and Richard Hull, "IFO: A Formal Semantic Database Model (Preliminary Report)," *ACM*, pp. 119-132, 1984.

2

U. Dayal and J. M. Smith, "PROBE - A Knowledge-Oriented Database Management System,"

On Knowledge Base Management Systems, Springer, New York, 1986.

3 U. Dayal and H. Wang, "View-Definition and Generalization for Database Integration in a Multidatabase System," *IEEE-Transactions on Software Engineering*, vol. SE-10, No. 6, pp. 628-644, November 1984.

4 Wolfgang Klas, Erich J. Neuhold, and Michael Schrefl, "An object oriented datamodel for a knowledge base," *this volume*.

5 *KnowledgeCraft User Manual*, Carnegie Group.

6 W. Litwin and A. Abdellatif , "Multidatabase Interoperability," *IEEE-Computer*, vol. 19, No. 12, pp. 351-381, IEEE Computing. Soc. Press, 1986.

7 W. Litwin, "Concepts for MultiDatabase Manipulation Languages," *4th Jerusalem Conference on Information Technology (proc.)*, pp. 309-317, IEEE Computing. Soc. PRess, Silver Spring, MD, USA, May 1984.

8 W. Litwin, "Implicit joins in the multidatabase system MRDSM," *IEEE-COMPSAC*, pp. 495-504, 1985.

9 Michael V. Mannino and Cynthia R. Karle, "An extension of the general entity manipulator language for global view definition," *Data&Knowledge Engineering*, vol. 1, pp. 305-326, North-Holland, 1985.

10 Amihai Motro, "Superviews: Virtual Integration of Multiple Databases," *IEEE Transactions on Software Engineering*, vol. SE-13, No. 7, pp. 785-798, July 1987.

11 Erich J. Neuhold, Wolfgang Klas, and Michael Schrefl, "Using object-oriented data base systems for modelling and representing multimedia objects," *internal circulated draft*, p. 20, Darmstadt, 1987.

12 E.J. Neuhold, "Objects and abstract data types in information systems," *Proc. of the IFIP TC2 Working Conference on Database Semantics; R. Meersmann, Steel T.B. (editors)*, pp. 1-12, North Holland, 1986.

13 E.J. Neuhold and M. Schrefl, "Towards Databases for Knowledge Representation," *Foundations of Knowledge Base Management. Contributions from Logic, Databases and Artificial Intelligence; Schmidt J.C., Thanos C. (editors)*, Springer, New York, 1987.

14 Michael Schrefl and Erich J. Neuhold, "A Knowledge-Based Approach to Overcome Structural Differences in Object Oriented Database Integration," *The Role of Artificial Intelligence in Database & Information Systems, IFIP Working Conf.*, Canton, China, July 88.

15 Michael Schrefl and Erich J. Neuhold, "Object class definition by generalization using upward inheritance," *Proceedings of the 4th International Conference on Data Engineering-IEEE*, p. 10, Darmstadt, 1988.

16 Amit P. Sheth, James A. Larson, and Evan Watkins, "TAILOR, A Tool for Updating Views,"

draft, October 1987.

17

Peter Wegener, "Classification in Object-Oriented Systems," *SIGPLAN Notices*, vol. 21, pp. 173-182, Oct 1986.

18

S.B. Yao, V.E. Waddle, and B.C. Housel, "View Modeling and Integration Using the Functional Data Model," *IEEE Transactions on Software Engineering*, vol. 8, No. 6, pp. 544-553, 1982.