

# A Graphical Yet Formalized Framework for Specifying View Systems

In *Advances in Databases and Information Systems, ADBIS'97*,  
Proc. 1st East-European Symposium, St.-Petersburg, Russia, September 2-5, 1997.  
Volume 1: Regular Papers. Nevsky Dialect, 1997, pp.123-132

Zinovy Diskin and Boris Kadish

Lab. for database design

Frame Inform Systems Ltd

Riga, Latvia

E-mail: zdiskin@acm.org, diskin@fis.lv      Fax: (371 7)828036

## Abstract

A graphical formalized language is proposed for specifying systems of views over database schemas. The language is based on the notion of arrow (mapping) between data schemas and is suitable for any data model for which schema mappings are defined. In particular, the constructs of query, query language, view and view integration can be consistently expressed in this arrow formalism and correspondingly specified. This gives rise to a general graph-based framework for specifying complex view systems. Basic constructions of the language and the entire framework as well can be considered as specialization of very general constructs developed in the mathematical category theory.

## 1 Introduction

The notion of view is one of the central ones in the database (DB) technology. Views make it possible to provide each application with its own presentation of data and isolate them from inessential (for them) details and changes of DB schemas. The practical importance of views is commonly recognized and, thus, a complex information system appears as a system of views over the DB, views over views, views over views over views and so on. In addition, these views can be somehow related between themselves, *eg*, one view can be the intersection of several other views, or the merge of several views, or both. In other words, the view system is itself a complex data structure subjected to integrity constraints and carrying certain operations. Hence, there is a practical need in languages specifying view systems in a compact and comprehensible but precise way.

The problem becomes especially actual in the context of modern information systems which are essentially heterogeneous and distributed over a DB net. As a result, the specification language in question should work even when views in the system are defined in different data models, *ie*, we need a language for specifying heterogeneous view systems.

The value of the issue is well recognized, however, while almost all is clear in the case of the relational data model, a proper notion of view against an OODB is still debatable. The situation is even worse in the context of heterogeneity, in fact, no sufficiently general notion of view was proposed, and each time the construct is managed in a *ad hoc* way (if any).

The goal of the present paper is to describe a specification language which makes it possible to specify heterogeneous view systems in a comprehensible yet formalizable way. Namely, we show that a *view* against a DB schema  $S$  is an arrow  $v: S_V \rightarrow \bar{S}$  which denotes a mapping from the *view schema*  $S_V$  into some augmentation  $\bar{S}$  of  $S$ ,  $\bar{S} \supset S$  with derived items. Of course, it is presupposed that for each given data model the notions of schema, derived schema and schema mapping are accurately defined, and so quite concrete metadata are hidden behind a view arrow as above. On the other hand, this allows to use the same arrow pattern for specifying view systems in different data models.

Thus, a view system in a given data model is described by a directed graph whose nodes are data schemas and arrows are schema mappings. In addition, constraints imposed on the system can be specified by declaring the corresponding properties for some diagrams in the view graph. So, a view system is normally specified by a *sketch* ([11]) rather than merely a graph, that is, by a graph in which some diagrams are labeled by predicate markers taken from some predefined signature. Different data models give rise to different marker signatures but this variety is not diversity: sketches in different signatures are nevertheless sketches, and they can be compared and integrated by special methods. Moreover, signatures are also specified by sketches and signature integration is reduced to sketch integration described in [8]. Anyway, in many practically interesting cases signature integration is not difficult.

The view-systems-as-sketches framework is polymorphic in its nature, that is, is suitable for any data model: one must only define what are schemas and their mappings. In particular, the framework can be applied to the sketch data model itself<sup>1</sup>, that is, both data and metadata may be specified by sketches. In the present paper sketches are essential on the metalevel and, simultaneously, are used on the level of data modeling to exemplify and demonstrate the essence of the approach.

One can see that the proposed framework is based on arrows and needs the corresponding arrow machinery (and even special *arrow thinking*). Such a machinery (and kind of thinking) were developed in the mathematical category theory<sup>2</sup> and have proven their extreme effectiveness in studying various systems of logic, in particular, logics employed in computer science and artificial intelligence. In a sense, the present paper can be considered as stating one more application of category theory in computer science. What we wish to stress in this respect is that abstract categorical constructs turned out to be unexpectedly close to the real software problem we discuss.

The rest of the paper is organized as follows. In section 2 a brief outline of sketches is presented to make the paper selfcontained. In section 3 the arrow setting for views is described. In section 4 a general graph-based framework for specifying federal database environments is presented.

**Acknowledgement.** We are indebted to Olga Tkacheva for careful TeXing (with Paul Taylor's "Diagrams") numerous sketches in this paper.

## 2 Data modeling via sketches

### 2.1 Sketches vs ER-diagrams: an example

The sketch data model was introduced in [11]. We remind that a sketch is a directed multigraph in which some diagrams are labeled with special markers. The intended semantics is to interpret nodes as sets and arrows as functions (both evolving in time) while a marker is interpreted as a certain (constant) constraint imposed on the diagram of sets and functions whose schema is labeled by the marker. In other words, nodes denote classes and value domains, arrows denote references and attributes, and markers denote integrity-checking procedures (a collection of diagram markers is presented in Table 1). In this way one obtains a variety of sketch data models by choosing one or another signature of markers.

Let us consider an example of semantic modeling via sketches. Suppose, the user is interested in information about men, married couples and married women of some town for, say, the last 50 years. Here "married women" means women which are or have been married during the last 50 years. A rough view on the universe is described by the ER-diagram on the top of Fig. 1 in a conventional ER-notation. Semantics of nodes and attributes is hopefully clear from its names, *MDate*, *DDate* are dates of marriage and divorce (the latter is optional). The domain of the optional attribute *Character* is a set consisting of three values *a, c, q*. The class *Woman* is of weak entity type since users are assumed to be interested only in women which are or have been married.

The sketch specifying the situation is depicted on the lower figure (see Table 1 for the meaning of the marked diagrams). Note the arc on the triple of arrows (*husb, wife, mdate*). It means that *Married*-objects are identified by their attributes *husb, wife, mdate*, the latter is necessary because a married couple can get divorced and then get married again. Note also monic markers on the key attribute *pin* and the cover marker on the arrow *wife*.

<sup>1</sup>application of sketches for semantic data modeling was described in [11, 10] and developed in [9]

<sup>2</sup>a standard reference suitable for computer science is [3]

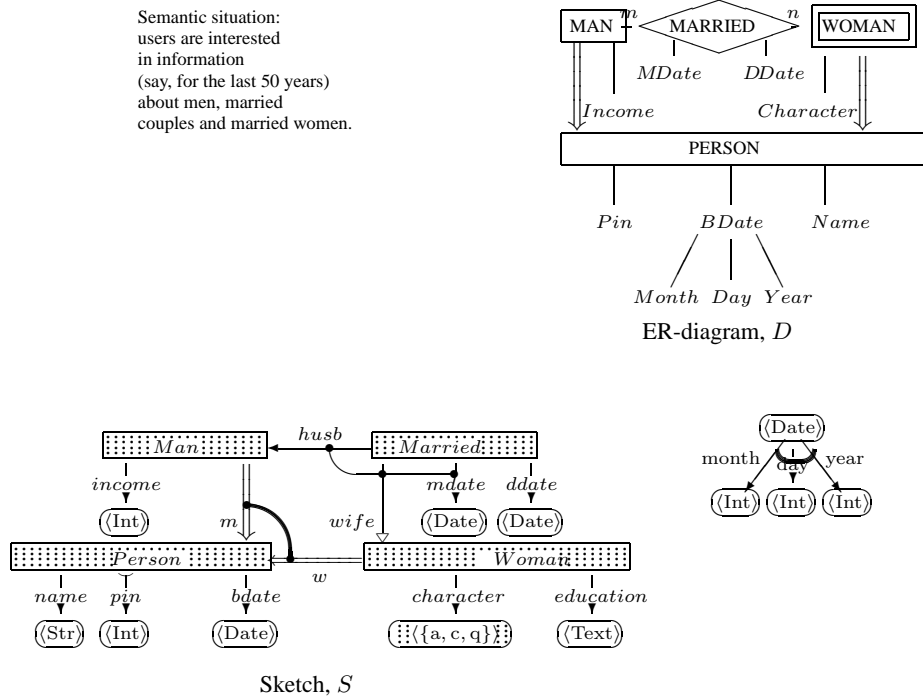


Figure 1: Semantic modeling via sketches

Rectangle nodes are *abstract classes* whose extensions should be stored in the DB: this is additionally pointed by small dots filling-in rectangles. Oval nodes are predefined *value domains* whose semantics is *a priori* known to the DBMS. For the sketch approach, `Int` and `{a, c, q}` are *markers* (in our precise sense) hung on corresponding nodes, that is, constraints imposed on their intended semantic interpretations. For example, if a node is marked by `Int` its intended semantics is the predefined set of integers.

## 2.2 Derived information via diagram operations

Consider, again, the simple universe described on Fig. 1. An important constraint that should be added to the schema is the condition of unique identification of any currently married couple by either its husband, or its wife as well. In other words, the subset of the relation ‘Married’ for which the attribute ‘*ddate*’ is undefined, is of the one-one relationship type. To express such a constraint, one should extend the original sketch with derived items as shown on Fig. 2: in order to distinguish basic items from the derived ones, the former are filled-in with small dots intended to remind about the extension to be stored. Of course, derived arrows should be also specially distinguished, and we agree to denote derived items by hanging various superscripts on their names (like  $'$ ,  $*$ ,  $\circ$  etc).

Certainly, the derived items of the sketch we consider can be obtained by an evident **Select-From-Where** SQL-query. We prefer, however, to specify this query as a composition of elementary diagram operations with sets and functions: *Null*, *ColImage*, *Composition*, presented in the top of Table 2 (as for the *Null* operation, we assume that each domain contains a single distinguished null value). Denotational semantics of operations is described in the corresponding column of the table.

In general, a diagram operation  $Q$  is specified by a sketch denoting its input data,  $S_Q^{\text{in}}$ , and a sketch denoting the output data,  $S_Q^{\text{out}}$ . It is convenient, however, to join  $S_Q^{\text{in}}$  and  $S_Q^{\text{out}}$  into one sketch  $S_Q$  so that an operation  $Q$  is specified by an inclusion  $\iota_Q : S_Q^{\text{in}} \hookrightarrow S_Q$  (in the Table 2 sketches  $S_Q$ ’s are called output sketches). The body of operation is then a procedure  $P_Q$  which calculates an extension of  $S_Q$  from a given extension of  $S_Q^{\text{in}}$ .

Thus, the derived items of the sketch  $\bar{S}$  on Fig. 2 can be presented as follows:

Table 1: A collection of diagram predicate markers (constraints)

Name	Arity Shape and Designation	Denotational Semantics
Separating Source		$(\forall x, x' \in X) x \neq x' \text{ implies } (\exists i) f_i(x) \neq f_i(x')$
Monic Arrow <sup>†</sup>	$X \xrightarrow{f} Y$	$(\forall x, x' \in X) x \neq x' \text{ implies } f(x) \neq f(x')$
ISA-Arrow	$X \xrightarrow{f} Y$ or $X \xrightarrow{f} Y$	$X \subset Y$ and $f(x) = x$ for all $x \in X$
Covering Flow		$(\forall y \in Y)(\exists i < n) y \in f_i(X_i)$
Cover <sup>†</sup>	$X \xrightarrow{f} Y$	$Y = f(X)$
Disjoint Images		$f_1(x_1) \cap f_2(x_2) = \emptyset$
Disjoint Covering Flow		$(\forall y \in Y \exists i < n) y \in f_i(X_i)$ and $i \neq j \text{ implies } f_i(X_i) \cap f_j(X_j) = \emptyset$

**define**  $(0', \{\perp'\})$  as *Null (Date)*  
**define**  $(CurrMarried^*, cm^*, dd^*)$  as *CoImage*  $(\{\perp'\}, ddate, 0')$   
**define**  $(husb^\circ)$  as *Composition*  $(cm^*, husb)$   
**define**  $(wife^\circ)$  as *Composition*  $(cm^*, wife)$

### 2.3 Composition of diagram operations (queries)

Queries can be composed by passing a part of the output data of a query (or a set of queries) to the input of another query operation. The way of passing is specified by the corresponding mappings of sketches. On the other hand, the composition can be presented as a stepwise augmentation of the initial schema with derived items. In fact, we have already used this procedure in the process of building the sketch on Fig. 2. A bit more involved example is presented on Fig. 4(a) text where we omit the fragment specifying derivation of items *CurrMarried*,  $cm^*$  described in Fig. 2.

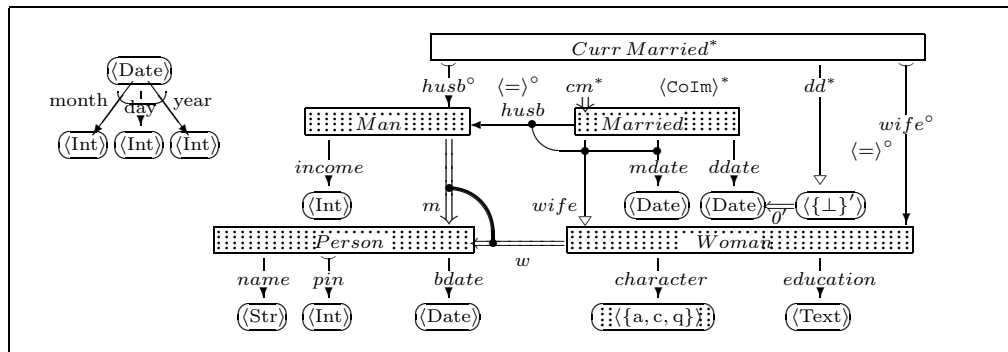


Figure 2: Sketches with operation markers: Specifying a query to a sketch = Extending it with derived items

Table 2: A collection of diagram operation markers (queries)

Name (marker)	Arity Shape		Denotational Semantics	Linear notation
	Input sketch	Output sketch		
Null	$\bullet A$	$A \xleftarrow{0} \{\perp\}$	$0(\perp) = \perp$	
Coimage (CoIm)			$B' = \{x \in X : fx \in B\}$ $f' = \text{restriction of } f \text{ on } B'$	$B' = f^{-1}(Y)$ or else $B' = \text{CoIm}_f(B)$
Image (Img)	$X \xrightarrow{f} Y$		$I = \{f(x) : x \in X\}$ $(\forall x \in X) f'(x) = f(x)$	$I = f(X)$ , or else $I = \text{Im}_f(X)$
Composition (=)			$(\forall x \in X) f(x) = g_2(g_1(x))$	$f = g_1 \circ g_2$
Projection (Proj) $_{\sigma}$ $\sigma: \{1 \dots k\} \subset \{1 \dots n\}$			$X' = \text{Im}_{p'}(X)$ where $p' : X \rightarrow Y_{\sigma(1)} \times \dots \times Y_{\sigma(k)}$ is set by $p'(x) = [f_{\sigma(1)}x \dots f_{\sigma(k)}x, ]$ $f'_j$ are projection arrows	$(X', f'_1 \dots f'_k) =$ $\text{Proj}_{\sigma}(X, f_1 \dots f_n)$
Pull-back (PB)			$R = \{(a, b) \in A \times B : fa = gb\}$ $p, q$ are projections	$R = \text{PB}(f, g)$
Difference (Dif)			$A' = \{x \in X : x \notin A\}$	$A' = X \setminus A$

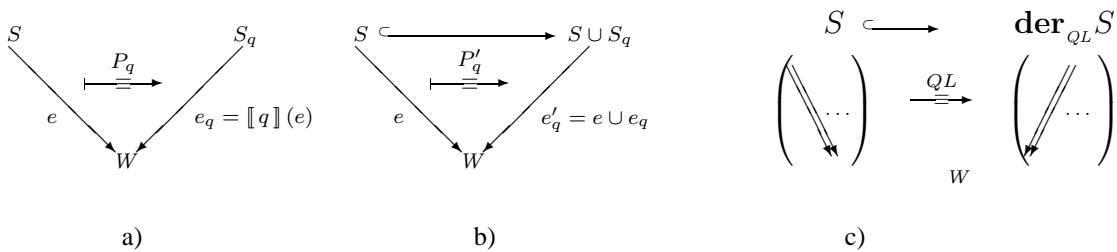


Figure 3: Queries via arrows

Given Table 2, with conventions adopted above the graphical image on Fig. 4(a) specifies a system of queries against the basic sketch in a unambiguous way. For example, the marker  $\langle \text{CoIm} \rangle^{\text{S}}$  labeling the the square diagram around it specifies that the extent of the node  $Man^{\text{S}}$  consists of those objects of the class  $Man$  for which the value of the attribute *income* is greater than  $10^6$ . Similarly, the extension of the node  $Man^{\text{I}}$  is the intersection of extensions of  $Man^{\text{S}}$  and  $Man^{\text{I}'}$  (note, the *CoImage* operation applied to two ISA-arrows turns into the ordinary intersection of sets). The node  $Man^{\text{I}'}$  is produced by the operation *Image* applied to the arrow  $husb^{\circ}$  which is derived, in its turn, by the composed query presented on Fig. 2. It follows from the semantics of these procedures that, *eg*, the extension of the class  $Man^{\text{I}'}$  consists of those happy  $Man$ -objects which are married (at the present time) and rich (with *income* no less than  $10^6$ ). Certainly, such a query can be expressed by standard SQL-means but the discussion of which of the languages is better is not relevant for our present considerations. All that we wish to demonstrate in this respect is how standard SQL-queries can be expressed by diagram operations over sketches.

## 2.4 Query languages: an abstract formulation

**2.4.1** The crucial observation is that semantic extensions of view schemas can be also described by arrows. If one thinks of a schema as a graph endowed with special marking, an extension of the schema is a mapping sending nodes to sets and arrows to functions (built over some predefined universe (world) of data objects,  $\mathcal{W}$ ) in such a way that intended semantics of the markers is respected. One can consider the collection of  $\mathcal{W}$ -sets and  $\mathcal{W}$ -functions as a graph whose nodes *are* sets and arrows *are* functions. In addition, given a marker (predicate)  $m$ , those diagrams of  $\mathcal{W}$ -sets and functions that possess the property denoted by  $m$  can be thought of as *marked* by  $m$ . In this way the universe can be converted into a (monstrous) schema  $W$  of the same type as semantic schemas we consider, and then an extension of  $S$  is nothing but a schema morphism (mapping)  $e: S \rightarrow W$ . The set of all possible semantic extensions of  $S$  will be denoted by  $\mathbf{Ext}(S)$ .

As it was said above, a query  $q$  is specified by its schema  $S_q$  together with a mapping

$$P_q = \llbracket q \rrbracket: \mathbf{Ext}(S) \rightarrow \mathbf{Ext}(S_q)$$

as shown on Fig. 3(a). One can add (derived) items of the schema  $S_q$  to the initial schema so that the query  $q$  is depicted by diagram Fig. 3(b). Thus, a query appears as an operation while the semantic schema  $W$  (in effect, the database in question) is a domain carrying this operation. This pattern is well known for the relational data model and can be generalized for semantic graph-based data models as well.  $\diamond$

**2.4.2** In many considerations it would be convenient to introduce a monstrous closure of a given schema  $S$ , which contains all possible derived items produced by QL-queries,

$$\mathbf{der}_{QL} S = \bigcup \{S_q \mid q \in QL\}.$$

Clearly,  $\mathbf{der}_{QL} S \supset S$  since any reasonable query language should contain trivial queries returning basic data without any operations on them. Certainly, for particular questions one needs only finite parts of  $\mathbf{der} S$ , however, the concept of the full closure can be very useful.<sup>3</sup>

In addition, any semantic schema (database) should be endowed with a mapping

$$\mu_W: \mathbf{der}_{QL} W \rightarrow W$$

corresponding to computing extents of derived items. Indeed, if an item  $X$  belongs to  $\mathbf{der}_{QL} S \setminus S$ , then it is a derived item and hence presents a (part of) query specification against the schema  $S$ . When  $S = W \in \mathbf{Sem}$ , the mapping  $\mu_W$  is available and then the item  $\mu_W(X)$  is the result of computing the query  $X$  over the database  $W$ . In fact, the mapping  $\mu_W$  makes the schema  $W$  closed under operations from  $QL$ .

So, in the abstract framework we suggest, a query mechanism is specified by a closure operator  $\mathbf{der}_{QL}$  over the collection of all possible schemas **Schema** such that databases can be considered as a  $\mathbf{der}_{QL}$ -closed objects of

<sup>3</sup>This is quite similar to the usefulness of the notion of the set of natural numbers which is nothing but  $\mathbf{der}\{0\}$  for the signature of operations consisting of a single operation *successor*

**Schema** . This is visualized by diagram Fig. 3(c) where the arrows in the bold brackets denote instances of schemas  $S$  and  $\mathbf{der}_{QL}S$  respectively.  $\diamond$

The considerations above can be formulated in abstract categorical terms by saying that the collection of all schemas is a category **Schema** and  $\mathbf{der}_{QL}$  is a monad over **Schema** while databases are algebras of this monad.

**2.4.3 Definition.** An *abstract data model* is a triple  $\mathcal{M} = (\mathbf{Schema}, \mathbf{der}, \mathbf{Sem})$  with **Schema** a category, **der** a monad (closure operator) over it, and **Sem** a collection of **der**-algebras (**der**-closed schemas).<sup>4</sup>  $\diamond$

Then a database instance over  $S$  is an arrow  $e: S \rightarrow W$  with  $W \in \mathbf{Sem}$  which can be extended in a unique way to the arrow  $\bar{e}: \mathbf{der}_{QL}S \rightarrow W$  such that the restriction of  $\bar{e}$  on  $S$  equals  $e$ .

### 3 Views via schema morphisms

**3.1 Getting definition.** There was a considerable debate in the database theory literature about the general notion of view against a semantic or logical database schema, in particular, in the object-orientation framework. Despite the large body of work done in the area ([1, 20, 19, 16] if to mention only a few recent publications), the notion of view was not precisely formulated (as we will show) and even the terminology is still somewhat confusing. Indeed, in the majority of works (like that just mentioned) the term *view* is used to denote derived items (as a rule, classes or relations), according to the pattern “**define view name as query specification**”. In some works, however ([22, 13]), a *view* to a schema is a subschema of the schema. An evident integrated formulation is to consider a view to a schema  $S$  as a subschema of some augmentation  $\bar{S}$  of  $S$  with derived items, ie,  $S_{View} \subset \bar{S} \supset S$ . This is an almost good definition but it forces one to consider the name space of view schema  $S_V$  as a subspace of that of the schema  $S$ . However, for the view user its schema  $S_V$  should be totally autonomous with its own name space.

Thus, a more correct consideration is to define a *view* over  $S$  as a pair  $V = (S_V, v)$  with  $S_V$  a schema (of the same kind that  $S$  is) and  $v$  a mapping (schema morphism)  $v: S_V \rightarrow \bar{S}$  sending items of  $S_V$  into those of  $\bar{S}$  (one may think of the items as names of nodes (classes) and arrows (attributes or references) ).

An essential advantage of this definition is that the mapping  $v$  is not bound to be one-one, that is, it can well be the case when two different items  $N, M$  in  $S_V$  are glued into one item  $K$  in  $S_V$ ,  $v(N) = v(M) = K$ . Then the presence of two different names in  $S_V$  could seem a trick that just misleads the view user, but note that the database schema  $S$  may evolve and later the item  $K$  may diverge into two different items  $K_1, K_2$  such that according to the new view semantics the view morphism  $v$  has to map  $N$  to  $K_1$  and  $M$  to  $K_2$ ,  $v(N) = K_1 \neq K_2 = v(M)$ . Or, vice versa, initially there were two different nodes  $K_1, K_2$  with  $v(N) = K_1, v(M) = K_2$  but then  $S$  has evolved to a state when  $K_1$  should be merged with  $K_2$ . Thus, by means of a suitable changing the view mapping  $v$  but *without changing the view schema*  $S_V$ , the DBA can conform an application built over the view with evolving database so that there is no need to rebuild the application.  $\diamond$

**3.2 Examples.** Let us again consider a simple conceptual schema, sketch  $S$ , on Fig. 1. Two simple views  $V_1, V_2$  on this sketch are presented on Fig. 4(b) on the left. Each of the views is specified by the view schema and a mapping which is set by the corresponding table; an augmentation of  $S$  required to set up these views is specified on Fig. 4(a). From these specifications it is seen, for example, that extension of the node  $PHusband$  in the view  $V_1$  consists of those  $S.Man$ -objects whose *income* is more than  $10^6$  and, simultaneously, they are not present in the relation  $\bar{S}.CurrMarried$ <sup>5</sup>. Indeed, the markers  $\langle Im \rangle'$  and  $\langle Dif \rangle \sim$  on Fig. 4(a) denote operations of taking the image of function and set difference respectively. Then the class  $Man \sim$  consists of those  $Man$ -objects which do not occur into  $CurrMarried$ <sup>\*</sup>-pairs. Further, the marker  $\langle CoIm \rangle^?$  defines the class  $Man^?$  as the intersection of  $Man \sim$  and  $Man$ <sup>s</sup>. The latter class is defined by the marker  $\langle CoIm \rangle^s$ , that is, it consists of  $Man$ -objects with *income* greater than  $10^6$  (so, extension of the node  $Man^?$  consists of rich unmarried men). According to the  $V_1$ -view mapping, the extension of the node  $S_{V_1}.PHusband$  is as was described above.

<sup>4</sup>Due to space limitations we omit explanation of the notions of monad and its algebras over a category which (unfortunately) do not belong to the collection of common use mathematical concepts. Note, however, that last time there were published several DB theory paper employing monads, eg. [4, 5].

<sup>5</sup>we will qualify sketch items by names of sketches if necessary

Similarly, extension of the node *Marr* in the view  $V_2$  consists of those  $\overline{S}.CurrMarried^*$ -pairs  $(w, h)$  for which  $w.character = a$  and  $h.income \geq 10^6$ .

Some delicate points associated with our definition of view are demonstrated in the right half of Fig. 4(b). A view mapping  $v$  must be a *schema morphisms*. That is, for the sketch data model,  $v$  is not merely a graph morphism but a sketch morphism: if a diagram  $D$  in  $S_V$  is labeled by a marker  $M$  then its image  $v(D)$  in  $S$  must be also labeled by  $M$ . For example, in the specification  $V_3$  on Fig. 4(b) the corresponding mapping is not a sketch morphism since the pair  $(S_{V_3}.husb, S_{V_3}.wife)$  is marked by an arc while its image, the pair  $(\overline{S}.husb, \overline{S}.wife)$  in the sketch  $\overline{S}$  is not labeled (well, the triple  $[\overline{S}.husb, \overline{S}.wife, \overline{S}.mdate]$  is separating but it does not imply that the pair is). Semantically, this means that according to the view schema  $S_{V_3}$ , any *Married*-object is identified by the pair of objects  $(wife, husb)$  whereas the data which the view extracts from the  $S$ -extension do not necessarily satisfy this condition. Hence,  $V_3$  is not a view on  $S$  in the technical sense.

The view  $V_4$  is a somewhat artificial example of view where the view mapping is not one-one (as for the arrow  $id_{Person}$ , we assume that any node in a sketch has the identity arrow into itself which is not depicted).  $\diamond$

**3.3 View metasketch.** The system of views we have just considered can be presented by a graph on Fig. 4(c). Moreover, this graph carries a sketch structure since view mappings and their diagrams are subjected to certain constraints. For example, the arc with brackets hung on the arrows  $v_1, v_2$  reflects the constraint that views  $V_1, V_2$  are disjoint. Also, views  $V_1, V_2$  are denoted to be one-one (see Table 1). One can see that specifying the sketch structure in the graph of views is important for capturing data semantics.  $\diamond$

**3.4 Semantics.** If  $\overline{S}$  is an augmentation of  $S$  with derived items, any  $S$ -extension  $e: S \rightarrow \mathcal{W}$  can be extended to an extension of the augmented schema,  $\overline{e}: \overline{S} \rightarrow \mathcal{W}$ , in a unique way as follows. If an item  $N$  from  $\overline{S}$  actually occurs into  $S$ ,  $N \in S$ , then  $\overline{e}(N) = e(N)$ ; however, if  $N \in \overline{S} \setminus S$  then  $N$  occurs into the schema of some query  $q$  against  $S$  and so  $\overline{e}(N)$  is the answer to  $q$  extractable from the set of data  $\{e(M) \mid M \in S\}$ .

Now, given a view  $v: S_V \rightarrow \overline{S}$  over a schema  $S$  and an extension  $e$  of  $S$ , the corresponding extension  $e_V$  of the view schema is the composition of two arrows:

$$e_V = v \triangleright \overline{e}: S_V \xrightarrow{v} \overline{S} \xrightarrow{\overline{e}} W$$

(the symbol  $\triangleright$  denotes the operation of arrow composition). Moreover, if  $v_1: S_{V_1} \rightarrow \overline{S_V}$  is a view over  $S_V$ , then the extension  $e_{V_1}$  of  $S_{V_1}$  is the composition

$$e_{V_1} = v_1 \triangleright \overline{v} \triangleright \overline{e}: S_{V_1} \xrightarrow{v_1} \overline{S_V} \xrightarrow{\overline{v}} \overline{S} \xrightarrow{\overline{e}} W$$

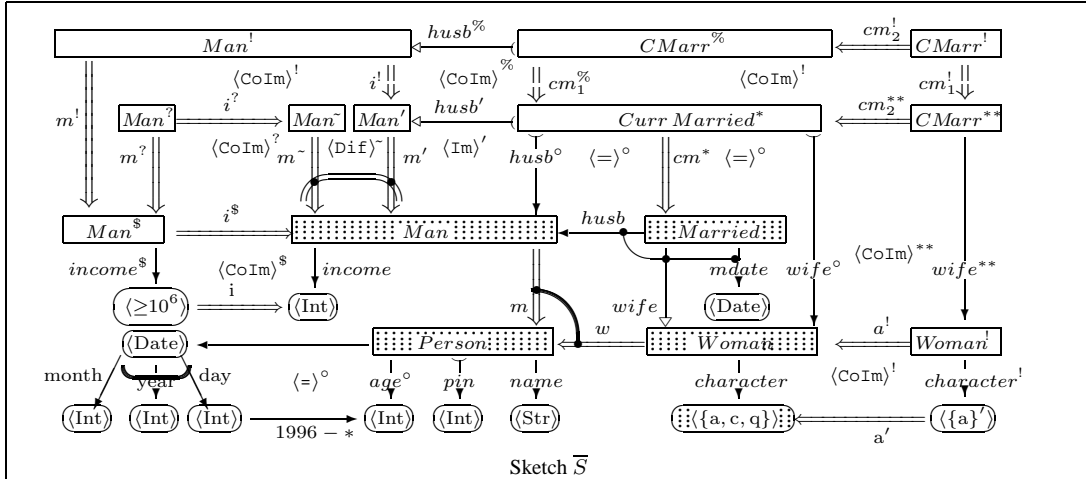
where  $\overline{v}: \overline{S_V} \rightarrow \overline{S}$ ,  $\overline{e}: \overline{S} \rightarrow \mathcal{W}$  are the corresponding augmented mappings.  $\diamond$

Thus, a system of views over a given schema is specified by a graph of view schemas and view mappings, and the well known mechanism of calculating view extension is specified by the arrow composition. In this way the view architecture can be described in a graph-based algebraic arrow framework.

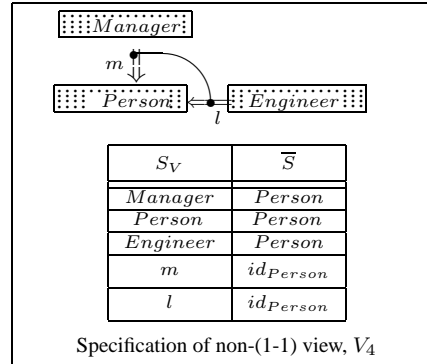
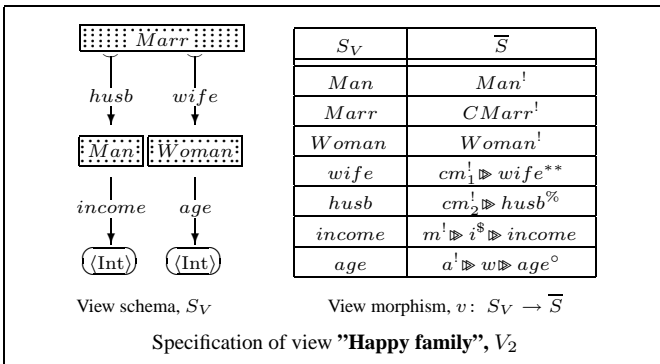
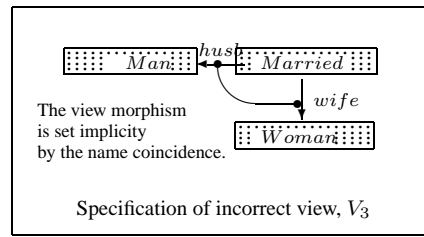
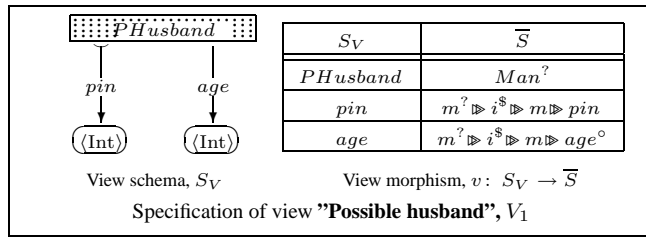
## 4 DB architecture schemas in the arrow formalism

**4.1** In the framework developed above, a centralized DB-system with a system of views over it can be specified by the diagram presented on Fig. 5 (we remind that the arrows in the bold brackets denote instances of schemas  $S$  and  $\mathbf{der}_{QL}S$  respectively). Note, the graphical image on the figure is not merely a picture convenient for heuristic discussion but also a precise formalized specification where each item (node or arrow) has its own semantic extent to be supported by the DBMS. One can think of the schema on Fig. 5 as displayed on the monitor's screen (of course, only a finite part  $\overline{S} \subset \mathbf{der}_{QL}S$  can and actually need be shown) and clicking an item will display the semantic content denoted by the item.

One can think of the node  $\mathbf{der}_{QL}S$  as a sufficiently large augmentation of  $S$  with derived items such that all view schemas  $S_i$  can be mapped into it (example of such an augmentation is presented on Fig. 4a). Arrows  $v_i$  are coupled



a) Composition of queries against the sketch S on Fig. 1



b) Several simple views to the sketch S on Fig. 1

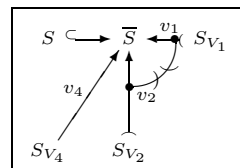


Figure 4: Specifying views over a sketch = Setting sketch morphisms into its augmentations with derived items

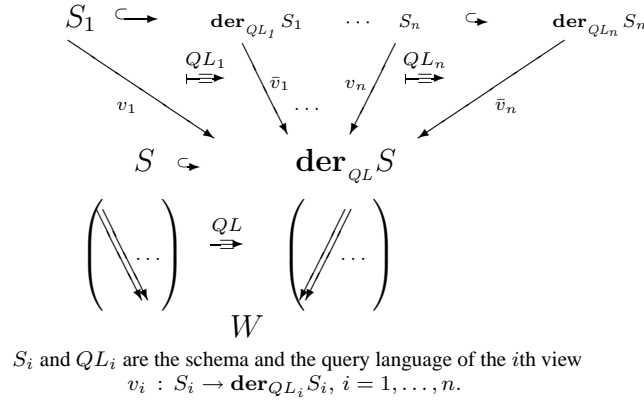


Figure 5: Meta-schema of a centralized DB with a system of views

with schema mappings stored in the view catalog (one can think of these mappings as tables similar to those presented on Fig. 4b). Arrows  $\bar{v}_i$  can be thought of as consisting of lists of procedures translating queries against view schemas into queries against the global central schema. All these data are metadata which form the syntactical side of the view system supported by DBMS.

On the semantic side, the arrow  $QL$  consists of the list of procedures  $P_q$  (see section 2.4) calculating  $QL$ -queries. Let  $e$  be an extent of  $S$  (a database instance) and  $q_i$  is a query against  $S_i$ . First,  $q_i$  is translated into a query  $q = \bar{v}_i(q_i)$  against  $S$  and then the procedure  $P_q$  returns the answer  $e_q$ , that is, a partially defined mapping  $\mathbf{der}_{QL} S \rightarrow W$ . It is then restructured into an answer to  $q_i$  according to the rules coupled with the translation  $\bar{v}_i$ .

The architecture and the view mechanism we have just described are well known and our formal treatment may seem superfluous detail. However, the benefits are in the clear separation of syntax and semantics, and in a compact transparent yet formalizable notation which gives rise to an easy-to-manipulate specification of the view system. In addition, the arrow presentation of the view mechanism makes easier the management of some of its subtle components by reducing them to compositions of arrows.  $\diamond$

**4.2** The same framework allows to specify formally the general architecture of federal DB systems substantially described in [21, 18] as shown on Fig. 6. The result is presented by the meta-schema (actually, a sketch!) depicted on Fig. 7: there are shown two federal supervises,  $a$  and  $b$  with integrated schemas  $S_a^I, S_b^I$  respectively, and a system of external views against the  $a$ -supervise (a similar system for  $b$  is not shown).

Correspondence information schemas  $S_a^{CI}, S_b^{CI}$  specify data about correspondence between component schemas. Federal supervise schemas  $S_a^I, S_b^I$  are obtained by integration of component schemas with the correspondence information schemas (an automated procedure of sketch integration in the presence of inter-schema conflicts was proposed in [6, 8]). The arcs hung on arrows coming into  $S_a^I, S_b^I$  denote the cover constraint (Table 1). They show that each item of the integrated schema can be found in either one of the component schemas or in the correspondence schema.

In the explanation column on the right of Fig. 7, "data model" means a triple  $\mathcal{M} = (\mathbf{Schema}, \mathbf{der}, \mathbf{Sem})$  as above, and ordinary arrows in the diagrams are morphisms in the corresponding category  $\mathbf{Schema}$ . In contrast, curly arrows in the "translation" rows are mappings between data models,  $\mathcal{M}_i \xrightarrow{\mathcal{M}}$  (cf. data model transformations studied by Kalinichenko in [15]). Conceptually, they are similar to the so called *institution morphisms* which have been studied in the institution theory, see [2] for references. Precise categorical description of data model mappings goes out the scope of the present paper and we restrict ourselves with a rough outline.  $\diamond$

**4.3** Let  $\mathcal{M}_i = (\mathbf{Schema}_i, \mathbf{der}_i, \mathbf{Sem}_i), i = 1, 2$  be two data models. Their mapping  $F: \mathcal{M}_1 \rightarrow \mathcal{M}_2$  consists of two components,  $F = (\phi, \delta)$ . The first one maps data schemas and their morphisms in the model  $\mathcal{M}_1$  into those of  $\mathcal{M}_2$ :

if  $S \in \mathbf{Schema}_1$  then  $\phi S \in \mathbf{Schema}_2$  and if  $h: S \rightarrow S'$  is a morphism in  $\mathbf{Schema}_1$  then  $\phi h: \phi S \rightarrow \phi S'$  is a morphism in  $\mathbf{Schema}_2$ . In addition, if  $W \in \mathbf{Sem}_1$  then  $\phi W \in \mathbf{Sem}_2$ .

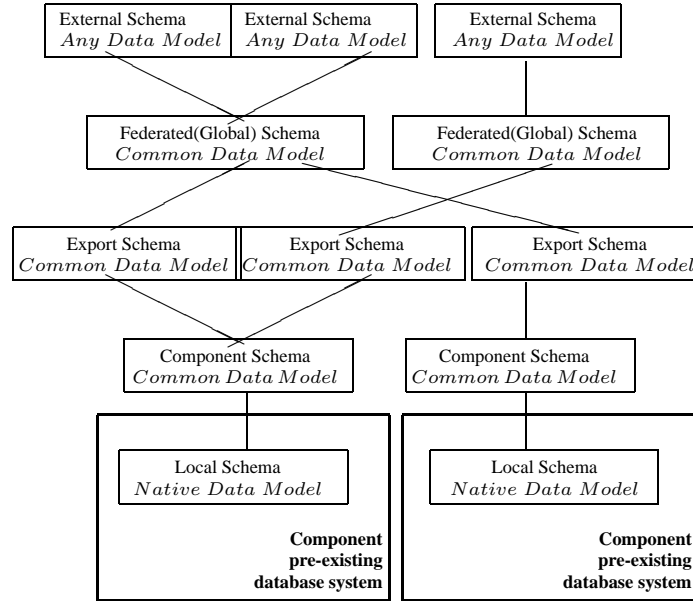


Figure 6: General architecture of a federal DB system (from [18])

These conditions provide that a database instance

$$e: S \rightarrow W, W \in \mathbf{Sem}_1,$$

in the first data model is transformed into a database instance

$$\phi e: \phi S \rightarrow \phi W \in \mathbf{Sem}_2$$

in the second model.

The second component of  $F$ ,  $\delta$ , is intended to explicate translation of  $\mathcal{M}_1$ -queries into  $\mathcal{M}_2$ -queries. That is, informally, if  $q$  is an  $\mathcal{M}_1$ -query against a schema  $S \in \mathbf{Schema}_1$  then  $\delta q$  is a  $\mathcal{M}_2$ -query against  $\phi S \in \mathbf{Schema}_2$ . Since this holds for any query, the entire  $\mathbf{der}_1$ -closure of  $S$  should be mappable into the  $\mathbf{der}_2$ -closure of  $\phi S$ . More formally,  $\delta$  is a mapping which sends any schema  $S \in \mathbf{Schema}_1$  to a  $\mathbf{Schema}_2$ -morphism  $\delta S: \phi(\mathbf{der}_1 S) \rightarrow \mathbf{der}_2(\phi S)$ .  $\diamond$

As a justification for the *abstract nonsense* we have just displayed, one can observe the nice similarity of Fig. 6 and Fig. 7 but the latter is a precise specification while the former is an informal picture.

## 5 Conclusion

On the technical side, the main result of the paper consists in abstract polymorphic definitions of data model (section 2.4) and view (section 3.1). They allow to specify architecture of view systems in a way possessing the advantages of being (i) graph-based, (ii) precisely formalized, (iii) polymorphic, that is, independent of any specific data model but capable to capture a wide class of them. The last property allows to manage specifications of heterogeneous view systems

Speaking in a broader context, the paper initiates building a data-model-independent framework for the DB theory. It seems to be a new research direction though important initial steps were made in a series of works by Kalinichenko ([14, 15]). The cause of the novelty is not that the issue was considered useless – contrariwise, its importance has been recognized for at least a few years, especially for the CoopIS area ([12]). One can guess that the actual cause

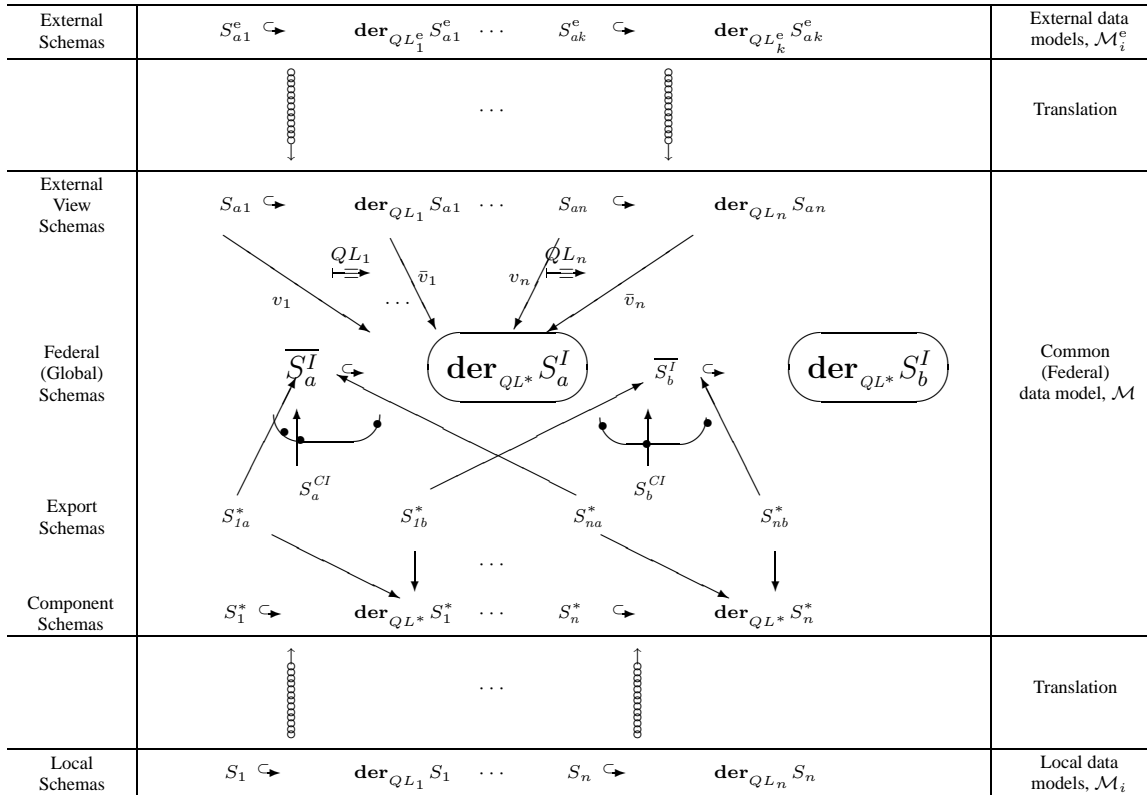


Figure 7: Meta-schema of a federal DB

is rooted in unfamiliarity of the community with appropriate specification tools. The present paper hopefully shows that the arrow machinery (and the style of thinking underlying it) developed in the mathematical category theory can be extremely suitable for stating the specification foundations of the problem. This gives rise to a difficult task of incorporating the arrow machinery into the IS design methodologies and techniques and, in a broader context, of inculcating the arrow style of thinking in the area of software engineering (cf.[7]).

Of course, the arrow thinking cannot be universally good for the entire field of information technologies. It is clear that in many situations string-based specifications are preferable, and often the common practice of using graphical schemas as a high-level informal language is quite relevant so that there is no need to employ graphical images as logical schemas (see a discussion in [17]). Actually, it is a question of experience and cognitive science research: when, where and to what extent employment of graphical schemas as formal logical specifications is effective and convenient for a human.

## References

- [1] S. Abiteboul and A. Bonner. Objects and views. In *Proc.ACM SIGMOD Conf. on Management of Data*, pages 238–247, 1991.
- [2] E. Astesiano and M. Cerioli. Relationships between logical frames. In *Recent trends in Data Type Specification*, LNCS No.655, pages 126–143, 1992.
- [3] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall International Series in Computer Science, 1990.
- [4] C. Beeri and P. Ta-Shma. Bulk data types. Theoretical approach. In *Proc.Int.Conf.on Database Programming Languages, DBPL'93*, 1993.
- [5] P. Buneman, L. Libkin, D. Suciu, V. Tanen, and L. Wong. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.
- [6] B. Cadish and Z. Diskin. Algebraic graph-based approach to management of multibase systems, I: Schema integration via sketches and equations. In *Next Generation of Information Technologies and Systems, NGITS'95*, 2nd Int.Workshop, pages 69–79, Naharia (Israel), 1995.
- [7] B. Cadish and Z. Diskin. Algebraic Graph-Oriented = Category Theory Based. Manifesto of categorizing database theory. TechReport 9506, Frame Inform Systems, 1995. ([//ftp.cs.chalmers.se/pub/users/diskin/MANIFEST/mnfst4.ps](ftp://ftp.cs.chalmers.se/pub/users/diskin/MANIFEST/mnfst4.ps)).
- [8] B. Cadish and Z. Diskin. Heterogenous view integration via sketches and equations. In *Foundations of Intelligent Systems, Proc. 9th Int.Symposium, ISMIS'96*, Springer LNAI'1079, pages 603–612, 1996.
- [9] Z. Diskin. Databases as diagram algebras: Specifying queries and views via the graph-based logic of skethes. Technical Report 9602, Frame Inform Systems, Riga, Latvia, 1996. (On ftp: [//ftp.cs.chalmers.se/pub/users/diskin/REPORTS/tr9602/\\*.ps](ftp://ftp.cs.chalmers.se/pub/users/diskin/REPORTS/tr9602/*.ps)).
- [10] Z. Diskin and B. Kadish. Variable set semantics for generalized sketches: Why ER is more object-oriented than OO. To appear in *Data and Knowledge Engineering*, manuscript is available by ftp [//ftp.cs.chalmers.se/pub/users/diskin/ER/ERvsOO.ps](ftp://ftp.cs.chalmers.se/pub/users/diskin/ER/ERvsOO.ps).
- [11] Z. Diskin and B. Kadish. Variable sets and functions framework for conceptual modeling: Integrating ER and OO via sketches with dynamic markers. In *OOER'95: Object-Oriented and Entity-Relationship Modeling*, Proc. 14th Int.Conf., Springer LNCS'1021, pages 226–237, 1995.
- [12] P. Drew, R. King, D. McLeod, M. Rusinkiewicz, and A. Silberschatz. Report on the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, 22(3):47–56, 1993.

- [13] S.J. Hegner. Pairwise-definable subdirect decomposition of general database schemata. In *3rd Symp.MFDBS'91*, Springer LNCS'495, pages 243–257, 1991.
- [14] L. Kalinichenko. Data model transformation method based on axiomatic data model extension. In *4th Int.Conf. on Very Large Data Bases, VLDB'78*, 1978.
- [15] L. Kalinichenko. Methods and tools for equivalent data model mapping construction. In *Advances in Database Technology - EDBT'90*, pages 92–119, 1990.
- [16] I.S. Mumick. The rejuvenation of materialized views. In *6th Int.Conf.CISMOD'95*, Springer LNCS'1006, pages 258–264, 1995.
- [17] M. Petre. Why looking isn't always seeing: Readership skills and graphical programming. *Communications ACM*, 38(6):33–44, 1995.
- [18] E. Pitoura, O. Bukhres, and A. Elmagarmid. Object orientation in multidatabase systems. *ACM computing surveys*, 27(2):141–195, 1995.
- [19] M. Scholl, C. Laasch, and M. Tresch. Updatable views in object-oriented databases. In *2nd Int.Conf.DOOD'91*, Springer LNCS'566, pages 189–207, 1991.
- [20] M.H. Scholl, H.-J. Schek, and M. Tresch. Object algebra and views for object bases. In *Int.Workshop on Distributed Object Management, Edmonton, Canada*, 1992.
- [21] A. Sneth and C. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 1990.
- [22] C. Tuijn and M. Gyssens. Views and decompositions from a categorical perspective. In *4th Int.Conf. on Database Theory, ICDT'92*, Springer LNCS'646, pages 99–112, 1992.