Heterogeneous View Integration via Sketches and Equations

In *Foundations of Intelligent Systems, ISMIS'96*,
Proc. of 9th International Symposium, Zakopane, Poland, June 9-13, 1996.
Z. Ras and M. Michalewicz (Eds.), Springer LNAI#1079, 1996, pp.603-612

Boris Cadish and Zinovy Diskin*

Frame Inform Systems, Riga, Latvia
E-mail: diskin@acm.org, diskin@frame.riga.lv

Abstract. In the paper a new approach to semantic modeling and view integration is proposed. The underlying data model is graph-based yet completely formalized so that graphical schemas themselves are precise specifications suitable for implementation: the approach is an adaptation of a familiar in the mathematical category theory specification framework based on the so called *sketches*. On this ground, a procedure of automated view integration is developed. Its distinctive feature consists in specifying correspondence between different views (on the same universe of discourse) by equations that reduces the integration task to a sequence of formal algebraic procedures.

1 Introduction and preliminary discussion

The term *view* (or, sometimes, *schema) integration* refers to the activity aimed at producing a global conceptual schema of an information domain from a set of locally developed user-oriented schemas (views). Integration is called *heterogeneous* if local views are specified in different data definition languages. View integration often appears to be a main component of conceptual design which is itself a part of the overall activity of software design. This explains the significant interest in schema integration methodologies: a vast diversity of various approaches, techniques and tools were proposed (see, *eg*, surveys [1, 11, 10]). Moreover, to date the value of the issue has increased greatly due to the tendency of organizing modern (and of the nearest future) information systems into federal environments where schema and data integration are among primary questions.

The main difficulty in view integration consists in managing structural conflicts between view schemas that occur when the same data are modeled differently in different views. Usually, semantic interpretations (extensions) of local schemas are overlapped, and due to possible different perception of the same

* Supported by Grant 93.315, and (the second author partly) by Grant 93.254 both from the Latvian Council of Science

piece of reality by different users the common part can be modeled by different constructs; the phenomenon is often called *semantic relativism*.[1] As a result, the global schema cannot be obtained by simple merging local schemas, and actually there is required a certain additional information (expressed in some language of *correspondence assertions*) about correspondence between local views.Moreover, searching for such a correspondence can reveal the necessity of introducing some new *interschema data* connecting local schemas but not captured by any of them. All this constitutes a heuristic and hard to formalize nature of the subject and leads to an abundance of *ad hoc* solutions and methodologies (see also [9]).

A lot of approaches to classifying and managing structural conflicts were proposed ([3, 1, 8, 12] and others), the most fundamental to date research is that one by Spaccapietra *et al* ([14, 13]) where a taxonomy distinguishing *semantic, descriptional, structural* and *heterogeneity* conflicts was suggested. However, the taxonomy appears to be an informal description rather than a precise specification capable to support automated integration. In general, while the phenomenon of semantic relativism is well known still there is no its formal explication, in other words, there is no a formal definition of what does it mean "the same data structured differently". Thus, semantic relativism is not explained in precise terms, and resolution of structural conflicts within a sufficiently rich data model is still a challenge. The more so is for heterogeneous conflicts where it seems little was done apart from stating the problem.

Our approach to the problem is based on consistent using two key ideas. The first one is to employ a graphical object-oriented data model possessing precise formal semantics[2]. It is a generalization of the *sketch* model worked out in mathematical category theory (CT) but it needs essential developing the standard sketch machinery (first steps were made in [4, 6]). Following the CT terminology tradition, we also call our graphical specifications *sketches*. The distinctive property of sketches crucial for handling heterogeneity is their provable universality: it can be shown that any formal data specification can be simulated by a sketch[3]. Thus, in our approach, all local views as well as inter-schema data are to be described by sketches so that integration of n views is reduced to inte-

[1] Moreover, a good semantic model must be rich enough to support relativism in order to supply each user with modeling constructs suitable to her perception. For example, in the ER data model the same object of the real world can be presented as an entity, or as an attribute, or as a relationship.

[2] We assert that none of the conventional graphical models has this property

[3] Any formal specification can be expressed in the language of some formal set theory, or in the language of some formal higher-order type theory as well (in fact, this is a definition of *formal specification* adopted in the modern mathematics). Then, there are three well known facts in CT: (1) set theories as well as type theories are interpretable in categorial structures called *toposes*; (2) toposes can be specified by sketches; (3) any topos can be presented as a (kind of nested relational) algebra over the corresponding graph.

The conclusion is that the full power of higher-order logic, including its algebraizability, can be simulated by sketches; or, in other words, *everything that can be specified formally can be specified by sketches*.

gration of $(n+1)$-sketches $S_1, \ldots, S_n, S_{CI}$ (where CI stands for *correspondence information*).

The second key idea consists in specifying structural conflicts via equations between algebraic terms. To wit, we show (see also [5, 2]) that all the diversity of structural conflicts can be uniformly described as follows: data considered as *basic* in one view are considered as *derived* in another view (derived from its own collection of basic data). This is nothing but an instance of the well known in mathematical logic phenomenon when basic operations, relations and even sorts of one many-sorted theory are not basic but derived in another theory. In the database context this means that data intended to be stored in the DB according to one view are considered to be retrieved (if requested) through corresponding queries against another view.

So, specification of structural conflicts is reduced to building augmentations $\overline{S_i}$ of local sketches with new items denoting derived information, $\overline{S_i} \supseteq S_i$, and then setting a set of equations between items of augmented sketches, E_{CI}.Then integration turns into disjoint merging local sketches, $(\overline{S_1} \oplus \ldots \oplus \overline{S_n} \oplus \overline{S_{CI}})$, and factorizing the result by the congruence generated by E_{CI}, that is, gluing together certain items of the merge according to the E_{CI}-equations.

A simple example demonstrating the idea is presented in section 3. In section 2 the machinery of data modeling via sketches developed in [7] is briefly described in the context of view integration.

2 Data modeling via sketches

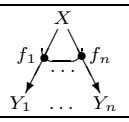
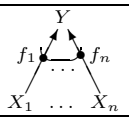
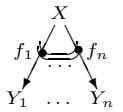
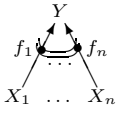
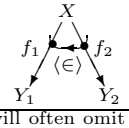
In the majority of conventional techniques for semantic modeling it is common to specify the intended semantic meaning of some node in a schema by marking the node with a corresponding label. In the context of view integration such a practice can readily lead to conflicts between views. For instance, in the standard of ER diagrams, to specify a node as a relation (say, $Marriage$), *ie*, as a set of relationship objects (marriage couples), one labels it by a diamond. However, if another user perceives the same data objects as entities (*eg*, families), (s)he labels the corresponding node in his schema by a rectangle. How must the node ($Marriage$) of the integrated schema be labeled?

One can observe that structural conflicts like above are caused by determining internal structure of a set via determining the structure of its elements: a relation is a set of tuples, a powerset is a set of subsets etc. In contrast to thinking in terms of elements, the category theory paradigm of arrow thinking suggests to specify internal structure of elements of a given set by characterizing (labeling) the corresponding diagram of functions adjoint to the set. Here is a very simple example: the sketch specification of relations.

We define *a source* to be a set X equipped with a family $\mathcal{F}$ of functions $f_i\colon X \longrightarrow Y_i$, $i = 1, ..., n$. Then one has a function $f = [f_1 \ldots f_n]$ from X into the Cartesian product $Y_1 \times \ldots \times Y_n$ determined in the standard way by setting $fx = [f_1 x, \ldots, f_n x]$. It is easy to see that f is injective, *ie*, X is actually a relation up to isomorphism, iff the family $\mathcal{F}$ satisfies the following *separation*

condition: $\forall x, x' \in X,\ x \neq x'$ implies $f_i(x) \neq f_i(x')$ for some f_i. So, to specify a set as a set of tuples one can safely leave the set itself without imposition of any constraints but constrain instead the corresponding source of outgoing functions to be separating. Correspondingly, on the syntax level, to specify a node as a relation one can safely leave the node without any label but mark instead the corresponding source of outgoing arrows by, say, an arc denoting the separation condition (in effect, this is nothing but a well known idea of designating a key of relation).

Similar arrow treatments of other conventional semantic constructs (*eg*, grouping) are presented in the following table in a hopefully self-explained way.

Name	Arity Shape and Designation	Denotational Semantics
Separating Source		$(\forall x, x' \in X)\ x \neq x'$ implies $f_i(x) \neq f_i(x')$ for some f_i
Monic Arrow	$X \overset{f}{\longmapsto} Y$	$(\forall x, x' \in X)\ x \neq x'$ implies $f(x) \neq f(x')$
ISA-Arrow or Inclusion	$X \overset{f}{\Longrightarrow} Y$ or $X \subset\!\!\overset{f}{\longrightarrow} Y$	$X \subset Y$ 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	$X \overset{f}{\longrightarrow\!\!\triangleright} Y$	$Y = f(X)$
Maximal Separating Source		$(\forall x, x' \in X)\ x \neq x'$ implies $f_i(x) \neq f_i(x')$ for some f_i, and $(\forall y_1 \in Y_1, \ldots, \forall y_n \in Y_n)$ $\exists x \in X$ s.t. $f_1(x) = y_1, \ldots, f_n(x) = y_n$.
Disjoint Covering Flow		$(\forall y \in Y \exists i < n) y \in f_i(X_i)$ and $i \neq j$ implies $f_i(X_i) \cap f_j(X_j) = \emptyset$
ϵ-relation		$(\forall y, y' \in Y_1)\ y \neq y'$ implies $\{f_2 x : x \in f_1^{-1} y\} \neq \{f_2 x : x \in f_1^{-1} y'\}$, *ie*, there is an embedding $Y_1 \longmapsto \mathbf{Powerset} Y_2$.

Following a tradition, we will often omit the marker of commutativity. So, $\langle = \rangle$ can be considered as the default marker. In contrast, the diagrams not assumed to be commutative are marked with puncture sign $\dashv$.

3 View integration via sketches and equations

It was demonstrated in the previous section that semantic modeling can be based on imposing conditions on arrow diagrams contrary to imposing conditions on nodes as is often done in current semantic schemas. It is remarkable that even the very replacement of labeling nodes by labeling arrow diagram allows to avoid

certain kinds of structural conflicts between views like that one described at the very beginning of the previous section (note, however, these are the conflicts to which a large body of works is devoted, see Spaccapietra *et al* [13]). But what are the real structural conflicts in the arrow approach?

We will expose the idea through considering a very simple example. Let us consider two ER-schemas presented on Fig. 1. It is clear that if the names 'People' in the first schema and 'Person' in the second one refer to the same class of objects in the real world, then the schemas are very much overlapped semantically. The question is how this can be expressed in a formal way.

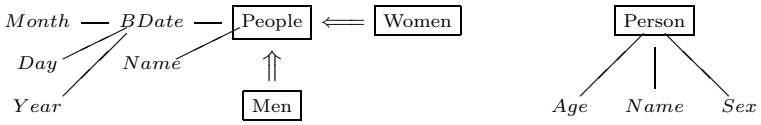


Fig. 1. Two ER-schemas to be integrated

A more thorough analysis of the situation shows that semantic overlapping amounts to the following: a part of the information considered as basic by the first view can be extracted from the second view as derived information (by making the corresponding queries), and vice verse. To specify this observation formally we proceed as follows.

First of all, we convert ER-schemas into sketches, *ie*, directed graphs some of whose diagrams are labeled by special markers - the resulting sketches are depicted on Fig. 2.

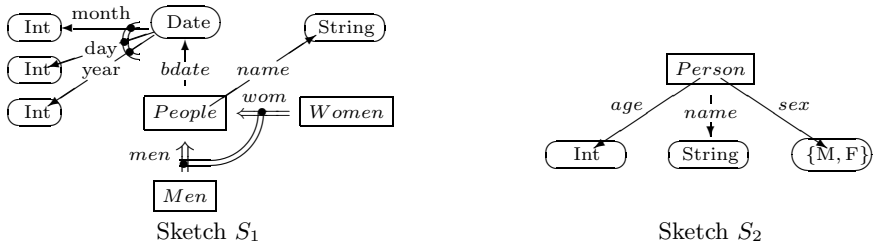


Fig. 2. Transformation of the ER-schemas into sketches

Note, that all attributes are presented by functions into nodes whose intended semantics is predefined (and supported by the computer system); in fact, these are predefined data types in the sense of programming languages (in particular, the domain of 'sex' consisting of two fixed tags, `M` and `F`, is a user-defined data type). In semantic modeling such nodes are usually called *printable* and are

specially labeled to distinguish them from *abstract* nodes. We label the former by stadium frames and the latter by rectangles.

REMARK. For us, `Int` and {`M,F`} are *markers* (in our precise sense), hung on corresponding nodes, that is, constraints imposed on their intended semantic interpretations. At the same time, 'Person', 'Men' etc. are merely *names* labeling corresponding nodes without imposing any constraints. Generally speaking, we should also give names to the nodes labeled by `Int` and {`M,F`}, say, 'Number' and 'Label'. However, since the intended semantics of so marked nodes is fixed and remains the same for all schemas (in any schema the intended semantics of a node labeled by the marker {`M,F`} is the two-element set consisting of the tag `M` and the tag `F`, despite the node's name: 'Label', or 'Tag', or 'Attribute' etc.), we adopt the convention of using such markers also as names; they will be printed in a non-italic font. So, abstract nodes are named, and printable nodes (value domains) are, in addition, labeled by markers expressing their intended semantics (while their names are omitted!).

Distinguishing between names and markers in semantic schemas appears to be a precise formalization of the well known differentiation between abstract and printable classes usually treated less formally.

Thus, semantic schemas are converted into sketches and we turn to specifying correspondence between views in the sketch language.

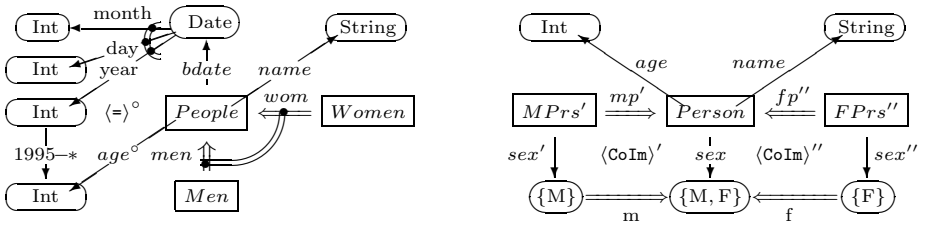
First of all, we make a default assumption that local sketches are defined over a common system of value domains and their names (markers!). The next step is to extend the original sketches with nodes and arrows denoting derived data so that correspondence between views should become explicit and could be formally described (Fig. 3). Specifically, to get derived elements on Fig. 3 we proceed as follows. [4]

To begin with, we extend the first sketch with a predefined function ($\mathtt{1995}-*$) of subtraction, and then compose three arrows, *bdate*, `year`,`1995-*` to get the arrow age° (and again, `year` and `1995-*` are markers rather than names). The marker $\langle\mathtt{=}\rangle^{\circ}$ labeling the corresponding diagram states the fact that age° is obtained by the operation of composition. As for the second sketch, we, first of all, enrich it with two inclusions, $\mathtt{m}\colon \{\mathtt{M}\} \hookrightarrow \{\mathtt{M},\mathtt{F}\}$ and $\mathtt{f}\colon \{\mathtt{F}\} \hookrightarrow \{\mathtt{M},\mathtt{F}\}$ whose existence is derivable from the existence of the set {`M,F`} .Then we apply twice the operation *CoIm* of taking, for a given function, the coimage of a given subset of the codomain in order to obtain the following derived items:

$$(MPrs', mp', sex') = CoIm(\mathtt{m}, sex), \quad (FPrs'', fp'', sex'') = CoIm(\mathtt{f}, sex).$$

The $\langle\mathtt{CoIm}\rangle$-marker labeling the two square diagram states exactly that the corresponding nodes and arrows are obtained by the diagram operation *CoIm* (and hence their extensions can be computed by the corresponding procedure).

[4] It would be convenient to picture initial sketches by one colour, say, black, and their derived components - by another colour, say, green. For typographical reasons, we replace green-painting derived items by hanging various superscript (like $'$,$''$,*, ,$^{\circ}$ etc.) on their names. In addition, derived nodes and arrows obtained simultaneously by applying some operation are marked with the same superscript, and the same superscript labels the very marker of the operation.



Sketch $\overline{S_1}$ Sketch $\overline{S_2}$

Sketches extended with derived items

(m,f denote respectively. inclusions of {M},{F} into {M,F})

$\overline{S_1}$	$People$	age°	$name$	Men	men	$Women$	wom
$\overline{S_2}$	$Person$	age	$name$	$MPrs'$	mp'	$FPrs''$	fp''

Correspondence equations, E_{CI}

Fig. 3. Specifying correspondence information

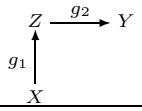
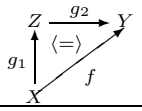
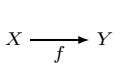
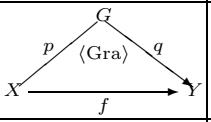
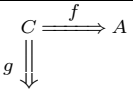
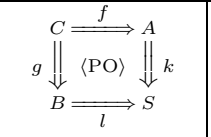
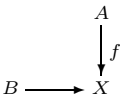
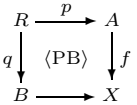
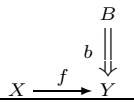
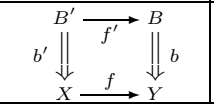
The sketch notation for graph-based operations is presented in Table 1. Each operation is specified by its *output sketch*, S_{out}, containing a designated *input* subsketch, S_{in}. Semantics is as follows: if an extension of S_{in} is given, then extensions of the output items belonging to $S_{out} \setminus S_{in}$ can be computed by the corresponding procedure. So, Table 1 presents a (small part of) graph-based query language.

Thus, we have extended initial black sketches S_1, S_2 to black-green sketches $\overline{S_1}, \overline{S_2}$ specifying also some additional (but necessarily derived) data. Now, the integrating person (a DB designer or administrator) can specify the correspondence between views in the form of equations which fix identification of the corresponding nodes and arrows. Namely, the required set of equations, E_{CI}, is presented in the table on Fig. 3 where two items standing in the same column give one equation, eg, $\overline{S_1}.men = \overline{S_2}.mp'$ *etc.* (Actually equality of two functions means, particularly, equalities of their domains and codomains respectively so that node equations in the table can be removed; further we will use the shortened notation). So, we have specified correspondence between views in a formal way and, moreover, our correspondence assertions are equations - this point has far reaching consequences.

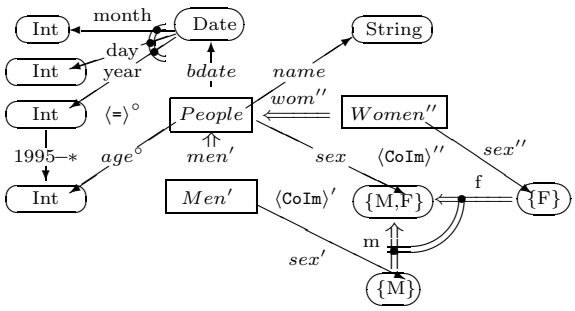
The next step of the integration process can be performed automatedly. It consists in gluing together those nodes and arrows that appear in the correspondence equations (in fact, this is nothing but taking the quotient of the disjoint sum of graphs by the congruence generated by the equations). The result of gluing is presented in Fig. 4; The name *full integrated sketch* refers to the fact that this sketch contains all the necessary information and, in addition, some derived (green) information.

To end the integration one must choose a *generating* subsketch S_I of the full sketch s.t. all items of $\overline{S_I}$ not contained in S_I can be extracted from the latter by means of queries, that is, diagram operations (Fig. 5). (Of course, there are

Table 1. A collection of diagram operations

Name ⟨**marker**⟩	Arity Shape: Input sketch	Arity Shape: Output sketch	Denotational Semantics	Linear notation
Composition ⟨=⟩			$(\forall x \in X) f(x) = g_2(g_1(x))$	$f = g_1 \gg g_2$
Graph ⟨**Gra**⟩			$G = \{(x, fx) : x \in X\}$ p, q are projections	$G{=}Graph(f)$
Push-Out (of ISA-arrows) ⟨**PO**⟩			$S = \{(a,1) : a \in (A \setminus C)\} \cup$ $\{(b,2) : b \in (B \setminus C)\} \cup$ $\{(c,3) : c \in C\}$	$S = PO(f, g)$
Pull-back ⟨**PB**⟩			$R = \{(a,b) \in A \times B : fa = gb\}$ p, q are projections	$R = PB(f, g)$
Coimage ⟨**CoIm**⟩			$B' = \{x \in X : fx \in B\}$ $f' =$ restriction of f on B'	$B' = f^{-1}(Y)$ $f' = f \lceil B'$

Following the tradition, we will often omit the marker of composition. So, ⟨=⟩ can be considered as the default marker. In contrast, the diagrams not assumed to be commutative are marked with puncture sign ⊹.



Full integrsated sketch (FIS), $\overline{S_I} = \overline{S_1} \oplus_{E_{CI}} \overline{S_2}$

Mapping $\sigma_1 : S_1 \to \overline{S_I}$

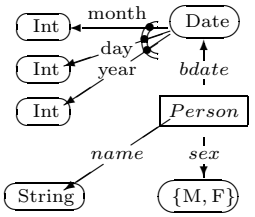
S_1	$bdate$	$name$	men	wom
$\overline{S_I}$	$bdate$	$name$	men'	wom''

Mapping $\sigma_2 : S_2 \to \overline{S_I}$

S_2	$name$	age	sex
$\overline{S_I}$	$name$	$age°$	sex

Mappings of local sketches into FIS

Fig. 4. Integration,I: Merging and Factorizing



Generating sketch,S_I

Mapping $\sigma_1 : S_1 \to \overline{S_I}$

S_1	$bdate$	$name$	men	wom
S_I	$bdate$	$name$	$m'.CoIm(sex,\mathrm{m})$	$f'.CoIm(sex,\mathrm{f})$

Mapping $\sigma_2 : S_2 \to \overline{S_I}$

S_2	$name$	sex	age
S_I	$name$	sex	$bdate;\mathrm{year};(1995-*)$

Mappings of local sketches into $\overline{S_I}$

Fig. 5. Integration,II: Removing redundant items

different generating subsketches; the order of preference is determined by psychological reasons like readability, or by readiness for future implementation, or by a combination of both). Together with a generating sketch, S_I, the procedure determines mappings $\sigma_i : S_i \to \overline{S_I}$ from local sketches into an expansion of the generating sketch with derived items.

In our example, to fix correspondence between views it was sufficient to state correspondence equations between basic and derived items of the initial schemas because correspondence consisted in *coincidence* of the corresponding items extents. However, in the general case, to specify correspondence equationally it is required to introduce new schema (sketch) fixing more complex correlations between extents than coincidence. For example, if the *People*-set of the first view is a subset of the *Person*-set of the second view, then one must introduce the specification presented by Fig. 6 and proceed with integration of S_1, S_2, S_{CI} along lines described above.

4 Conclusion

The example we have described is very simple yet demonstrates the general strategy for heterogeneous view integration we suggest. Every particular situation needs elaboration of the appropriate specific signatures of diagram markers

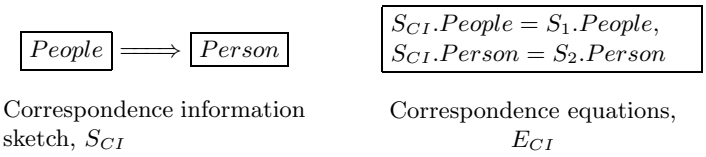


Fig. 6. Another example of specifying correspondence information

and operations similar to those described in Tables 1,2. After that is done, integration can be performed along lines we described.

The essence of resolving structural conflicts between view schemas consists in *discovering* suitable augmentations of local schemas (sketches) with derived items so that their correspondence could be described by equations. As a result, several extensive integration steps can be reduced to formal algebraic manipulations with terms and equations that provides their effective computer realization.

Actually the integration procedure is performed in two steps. The first one consists in disjoint merging $(n + 1)$ graphs underlying local schemas and then factorizing the merge (this step can be performed in a fully automatic way). The second step consists in converting the integrated graph into a schema by integrating diagram markers from the local sketches. However, here marked diagram conflicts can arise (we avoided them in our very simple example), and these are real conflicts between views which can bring to light their inconsistency (see [4, 6] for details). Interestingly, it seems that just this kind of conflicts were not identified in the DB theory literature.

Clear semantics of sketches makes it possible to distinguish in the general integration procedure the steps which can be fully automated and those that need human assistance. In its turn, the latter can be also made computer-aided and, speaking in general, the sketch framework brings to light limitations and possibilities of automation in schema and data integration(see also [2]). On the other hand, our experience has shown that sketches turn out to be very handy as a machinery for semantic modeling and integration. In particular, they are convenient for semi-automated phases of integration being performed in a interactive mode.

References

1. C. Batini, M. Lenzerini, and S. Navathe. A comparitive analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
2. 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 InfoTechnologies and Systems, NGITS'95*, 2nd Int.Workshop (Israel), 1995.
3. U. Dayal and H. Hwang. View definition and generalization for database integration of a multibase system. *IEEE Trans. Software Eng.*, 10(6):628–644, 1984.
4. Z. Diskin. Mathematical aspects of schema integration. TR-9502, Frame Inform Systems, Riga, 1995. (On ftp: //ftp.cs.chalmers.se/pub/users/diskin/tr9502.*).

5. Z. Diskin. Formalizing graphical schemas for conceptual modeling: Sketch-based logic vs. heuristic pictures. TR-9501, Frame Inform Systems, Riga, 1995.
6. Z. Diskin and B. Cadish. Databases as graphical algebras. (On ftp: //ftp.cs.chalmers.se/pub/users/diskin/amast96.*), 1995.
7. Z. Diskin and B. Cadish. Variable sets and functions framework for conceptual modeling: Integrating ER and OO via sketches with dynamic markers. In *Proc. 14th Int.Conf. (OO & ER)'95*, Springer LNCS'1021, 1995.
8. J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Trans. Software Eng.*, 15(4), 1989.
9. S. Navathe. The next ten years of modeling, methodologies and tools. In *ER'92*, number 645 in LNCS, (Karlsrue, Germany), 1992.
10. E. Pitoura, O. Bukhres, and A. Elmagarmid. Object orientation in multidatabase systems. *ACM computing surveys*, 27(2):141–195, 1995.
11. A. Sneth and C. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 1990.
12. S. Spaccapietra and C. Parent. Conflicts and correspondence assertions in interoperable databases. *ACM SIGMOD Record*, 20(4):49–54, 1991.
13. S. Spaccapietra, C. Parent, and Y. Dupont. Model-independent assertions for integration of heterogeneous schemas. *Very Large Databases Journal*, 1(1), 1992.
14. S. Spaccapietra, C. Parent, and Y. Dupont. View integration: a step forward in solving structural conflicts. *IEEE Transactions on KDE*, 1992.