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*,
Proc. 2nd Int.Workshop, Naharia, Israel, June 1995.
A. Motro and M. Tennenholtz (Eds.), 1995, pp.69-79

Boris Cadish and Zinovy Diskin*

Frame Inform Systems,
Elizabetes Str. 23,
Riga, LV-1234, Latvia
diskin@frame.riga.lv

1 Introduction and motivating discussion

Now it is evident that a chief property of the next generation information systems is their organization on cooperative principles. In its turn, for cooperative systems (CIS) it is normal that an application program needs data stored in several separate databases (DBs). In fact, as is was noted in [Mot87], such a *multibase situation* is very similar to the *multifile situation* before invention of DBs, and a natural solution is analogous: in order to provide applications with a single integrated view of data, local DBs should be integrated into a distributed DB. The later is a database system which has a schema as an ordinary DB but its extension is *virtual*: it is not stored but can be computed if requested. In addition, a peculiarity of CIS consists in heterogeneity of local DB systems caused by orientation on different level applications, and/or by the autonomy of their origin and initial development. So, heterogeneous multibase integration appears as a fundamental issue of CIS functioning. In particular, in the context of federated database systems (FDBS) , integration is a function regularly performed at different levels and by different services depending on the organization of the FDBS environment.

Significance of the problem is well known, various approaches, techniques and sometimes tools were proposed (see, *eg*, [DH84, Mot87, WHW90, YAD+92, SPD92b, SPD92a, SST92] and surveys [BLN86, SL90]). In spite of the diversity of approaches, several common points can be well identified.

Integration consists of *schema integration* – composing a global schema from the set of local ones, and *data integration* – computing virtual extension

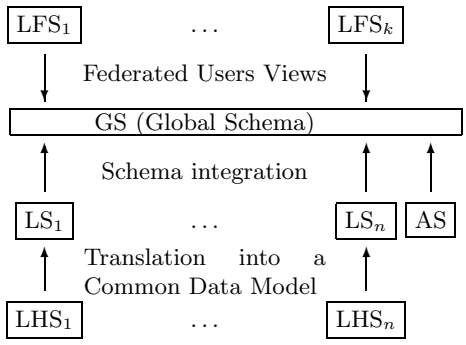


Figure 1: Schema integration architecture

of the global schema. In practice this means setting a collection of procedures which convert any query against the global schema into a set of queries against local schemas and then compute the global query answer by summarizing the local answers. Schema integration is the key issue: its correctness determines the correctness and effectiveness of the second phase. The typical general architecture was described already in [DH84] and remains in essence the same (cf. [SL90, YAD+92]) as presented on Fig.1 Local host schemas (LHS's) can be specified in different DDLs. An auxiliary DB with the schema AS may be needed to record information required for integration. The global schema (GS) is defined as a superview of LSs and AS. Its definition presupposes resolving various conflicts between local databases so that GS provides users with the illusion of a homogeneous and integrated DB. In addition, different (federated) views against GS may be defined for different applications.

In some approaches (*eg*, [DH84, Mot87] and others), schema integration is performed by consecutive applying structural operations from a certain predefined collection to the component schemas. Syntactically these operations are usually specified by a special non-procedural view definition language.

*Supported by Grant 94.315 from the Latvian Council of Science

In other approaches (developed mainly by Spaccapietra *et al*), correspondence between local databases is specified by a set of assertions and then the global schema must be built automatically. In this case a special *language of correspondence assertions* should be specified.

However, despite the large body of work done in the area, several fundamental questions remain unanswered, moreover, often they are not stated at all.

For example, in different integration methodologies and tools different data models are proposed to serve as a common data model. However, the question whether a given data model can be really universal, or to what extent it is universal, or how natural translations from conventional data models into this common data model *etc* are usually out of scope. Similarly, it remains open whether a given view definition language , or a correspondence assertions language are sufficient for specifying the variety of (all?) possible situations of local DB overlapping.

The last question is immediately connected with an extremely important issue of classifying conflicts between local schemas in their representation of the overlapping information. Indeed, it is well known that the same data can be structured in different ways, and resolving such structural conflicts between representations is the key for correct integration. In fact, a methodology of integration is mainly determined by the treatment and the taxonomy of conflicts it adopts. Several approaches were proposed ([DH84, LNE89, SP91, SPD92b] and others), the most fundamental to date research is [SPD92a] where a taxonomy distinguishing *semantic, descriptional, structural* and *heterogeneity* conflicts was proposed. It appears however that this is rather an informal description of various intuitions behind conflicts (and, indeed, it properly encompasses the diversity) than a precise formal specification capable to support automated integration.

Speaking in general, it appears that 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 setting the problem and several declarations.

It is evident that answering the bundle of (methodological but of an immediate practical impact) questions mentioned above is possible only in a formal semantics framework for semantic modeling and integration. The lack of such a framework is commonly recognized , an illuminating example is the fact of invoking abstract Wittgenstein's philosophy on a conference organized by practitioners([DKM$^+$93]), in the conclusion report of which it was emphasized that there is a great need in languages for specifying data interoperation.

In fact, nowadays a database designer is forced to alter between *Scylla* of easy-to-use but scarcely formalized graph-based approaches, and *Charybdis* of logically clear and perfect but inflexible relational languages. Due to the evident tendency of any practitioner towards the former, what exists out in the field today is "cookbook approaches" poorly formed and having too many subjective human considerations (Navathe [Nav92]) so that specifications are mixed with implementations and a procedural description is often the problem statement and its resolving algorithm simultaneously . Numerous attempts of DB theorists to develop an appropriate machinery seem to be not very successful in that the proposed approaches are not completely formalized or/and are too complicated to be adaptable for practical using (especially in comparison with clear and concise relational data model) .

This is not surprising: while the relational data model is based on the firm mathematical ground of first/higher order logics (FOL/HOL), no such a graph-oriented mathematical foundation is familiar to the DB theory community. In addition, the subject is rather involved and, we would like to emphasize this, is hard to be developed in a naive way supported mainly by the common sense without mathematical ground. Indeed, graph-based specifications is the focus of the mathematical category theory [1] whose experience showed that the subject requires a special kind of thinking – *arrow thinking* – contrary to usual mathematical thinking in terms of sets and elements. Our revision of conventional semantic modeling constructs from this view point has shown that tackling graph-based specifications in a naive way (without careful respecting some delicate points category theory teaches us) can lead to awkward or/and irrelevant constructions, confusing terminology, ambiguities and quasi-formal *ad hoc* considerations.

On the other hand, the revision has demonstrated that *all* necessary notions and constructs are actually familiar to the community but in a intuitive and often implicit form. So, all that is required is to bring more mathematical order in the subject and organize a weakly connected collection of constructs into an integrated consistent framework. [2].

In this paper we propose a graph-based specifica-

[1] To date there are several books oriented on general Computer Science applications, see, *eg*, [BW90]

[2] The present paper is written by a specialist in DB design and a mathematician, and our collaboration convinced us in an extremely high relevance of categorical logic methodology and machinery for stating the methodology and machinery for heterogeneous schema integration, and building the DB theory as a whole ([CD94, CD93, **?**, Dis92]).In fact, nowadays DB theorists often recover main concepts of category theory, and during the last 3-4 years there appeared several publications on applying category theory in the DB area. However, their considerations are somewhat orthogonal to the approach we suggest (while the suggestion in [**?**]) is parallel!).

tion language with rigorously formalized semantics so that the very graphical images can be considered precise formal specification(like, *eg*, FOL theories). A similar idea was developed in category theory through the concept of *sketch* (a standard reference is [BW90]). Actually, our framework is a sketch-like generalization of the ordinary FOL formalism (normally supporting mathematical structures over sets) for the case of structures over graphs .

A great advantage of the sketch specification language is that it is (provably!) universal: it follows from general results of categorical logic that any formal data specification [3] can be replaced by an equivalent sketch of some appropriate type. So, sketches provide a real possibility to handle heterogeneity in a consistent (and effective, as we see below) way.

In addition, rigor semantics of sketches makes it possible to distinguish in the general integration procedure those parts which can be fully automated and those that have to be performed by an integrating person (DBA). In its turn, the latter phases can be also made computer-aided and, on the whole, the sketch framework brings to light limitations and possibilities of automation in schema and data integration.

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. In addition, since the sketch treatment of integration is essentially algebraic several extensive integration steps can be reduced to formal algebraic manipulations with terms and equations which provides their effective computer realization.

Thus, the approach we suggest provides the real possibility of automated heterogeneous integration. We call it *AGO* – an abbreviation of Algebraic-Graph-Oriented.

The AGO-integration is outlined in the next section. In section 3 we present several concrete examples of semantic modeling via sketches, in particular, familiar aggregation, generalization and grouping constructs. A simple demonstrating example of schema integration is considered in section 4.

2 Outline of the approach

1 Sketches. The core of AGO is a family of graph-based specification languages in which specifications are directed multigraphs endowed with a labeling formalism of a special kind. As was said, we call AGO-specifications *sketches* following the terminology tradition of categorical logic (however, AGO-sketches are much more general than the standard category theory sketches). [4]

The essence of the sketch approach to data modeling consists in considering all classes of objects homogeneous while all type information about objects is contained in arrow (in particular, attribute) structures connected with classes. [5]

Formally, sketches are graphical constructs consisting of three kinds of items: (i) nodes, to be interpreted as sets, (ii) arrows, to be interpreted as functions between corresponding sets, and (iii) diagram markers, to be interpreted as constraints imposed on diagrams labeled by these markers (under a diagram in a graph G one could mean a certain collection of nodes and arrows of G).

Following well known techniques of semantic modeling, AGO treats the distinction between objects and values via distinguishing between abstract and printable classes which, in its own turn, is also captured by special discipline of labeling. This provides the possibility to combine class and type hierarchies in a single sketch.

2 Sketch operations vs conflicts. For AGO, all kinds of conflicts distinguished in [DH84, SP91, SPD92a] and elsewhere are special cases of the well known distinction between basic and derived information: data considered as basic in one view (and actually stored in a database) can be safely considered as derived (from another collection of basic data) in another view (and thence actually must be computed if requested). The essence of resolving such conflicts consists in search of operations (queries) transforming data from one presentation into another. In particular, for AGO the process of conforming view schemas (for further integration) amounts to extending them with additional items denoting derived information *so that the correspondence between views could described by equations.*

For database integration, each of the sketch-extending operations must be coupled with a procedure computing extensions of the derived items produced by the operation. Indeed, from a computational view point, the output shape of an operation is nothing but a graph-based specification of the corresponding query, and the procedure assigned to the

[3] *ie*, such a specification which, in principle, can be written down by a conventional formal language of the modern mathematics

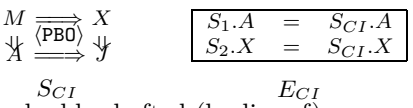
[4] Well,semantic schemas are also graphs endowed with special labeling , however, the question is which graphs should be used and what is to be labeled. A distinctive property of AGO-sketches providing essentially all their advantages just consists, firstly, in the choice of directed multigraphs as carriers of semantic schemas and, secondly, in a rigorously formalized and consistent discipline of their labeling.

[5] Note, merely removing type information from nodes to arrow diagrams prevents appearance of some kinds of structural conflicts, *eg*, like those that arise in ER-modeling when a class is represented by an entity node in one view schema and by a relationship node - in another.

operation compute the answer to this query.

3 Correspondence information via equations. In AGO, view correspondence information, including interschema data, is specified in a fully formal language, moreover, the language is essentially equational.

Actually, specification of the correspondence information is reduced to an additional sketch, S_{CI}, together with a set of equations of a special kind, E_{CI}. For example, if an AGO-integrator is informed that a node A (basic or derived) from one sketch, S_1, and a node X (basic or derived) from another sketch, S_2, are semantically overlapped [6] and their data extensions are intersected at any moment of time, the integrator builds the following *correspondence information specification*:



Here double shafted (bodies of) arrows denote inclusions: in fact, these are markers hung on arrows and constraining semantic interpretations of these arrows to be not merely functions but inclusions. The symbol $\langle \texttt{PBO} \rangle$ is another marker constraining the whole diagram to satisfy the condition $M^{\varepsilon} = X^{\varepsilon} \cap A^{\varepsilon}$, $J^{\varepsilon} = X^{\varepsilon} \cup A^{\varepsilon}$ where $M^{\varepsilon}, N^{\varepsilon}, X^{\varepsilon}, A^{\varepsilon}$ denote extensions of the corresponding nodes.

In general, the set of correspondence equations may contain arrow as well node equations expressing assertions 'two nodes/arrows are equal' whose semantics is evident. The initial sketches to be integrated are just enriched with derived items in a way providing the possibility of an equational description of the correspondence.

So, the only correspondence assertions considered in AGO are equations. It is remarkable that general results of categorical logic guarantee that *any* situation of schema integration which can be specified formally can be also specified in the above described way.

4 Integration and marker conflicts. From the formal view point, the sketch S_{CI} is similar to local schemas so that integration of n local schemas turns into disjoint merging $n+1$ sketches, $S_1, \ldots, S_n,\ S_{n+1}$ with $S_{n+1} = S_{CI}$, and factorizing the result by the congruence generated by E_{CI} (*ie*, gluing together certain items of the merge according to the E_{CI}-equations).

Quasi-formally, the result of the above-mentioned manipulations can be described as

$$\overline{S_I} \ = \ (\overline{S_1} \oplus \ldots \oplus \overline{S_n} \oplus \overline{S_{CI}}) \ / \ E_{CI}$$

where $\overline{S_i}$'s denote results of extending local sketches with derived items .

[6] We do not worry about how such a knowledge could be obtained

Actually this procedure is performed in two steps. The first one consists in disjoint merging $(n + 1)$ graphs underlying local sketches and then factorizing the merge according to E_{CI}-equations (this step can be performed in a fully automatic way). The second step consists in converting the integrated graph into a sketch integrating diagram markers from the local sketches. However, here diagram conflicts can arise, and these are real conflicts between views which can bring to light inconsistency of views!

Relation to other integration methodologies. Conceptually, AGO-sketches are close to the well known family of Functional Data Models ([Shi81, DH84, AH87, HK87]) but are distinguished by a special approach to labeling. In AGO the functional approach is systematically developed, precisely formalized in the mathematical sense and enriched with the machinery of sketch operations.

In its general integration strategy AGO is close to the superview approach described in [DH84, Mot87] but differs seriously in its realization. In particular, AGO view definition and restructuration language is specified in terms of sketches and sketch operations which provides formal semantics and algebraizability of the integration procedure.

In respect to the taxonomy of conflicts and correspondence assertions proposed in [SP91, SPD92a], it can be said that AGO reduces all that conflicts to conflicts of two kinds - structural conflicts (of being basic/derived) and constraint conflicts (between diagram markers), while all kinds of correspondence assertions are reduced to equations between arrows and equations between nodes. Among these conflicts, only marker conflicts are responsible for consistency of views and actually need resolution (of which of conflicting markers is dominant). In particular, there are possible situations when conflicting markers are not compatible.

In contrast, structural conflicts are always solvable in the sense that different structural representations of the same data are compatible but integration needs precise specification of the correspondence between them (so that the focus is *to discover* appropriate operations for extending local schemas with derived items). From this view point it would be reasonable to name structural conflicts by a different term, say, *structural distinctions*. Thus, the AGO-vocabulary of concepts for view integration is as follows:

- *signature of diagram markers* (constraints),
- *signature of diagram operations* (queries),
- *local sketches, correspondence sketch*,
- *extended sketches* (via application of operations),
- *correspondence equations*,
- *structural distinctions* (in being basic/derived),
- *constraint conflicts* (between diagram markers).

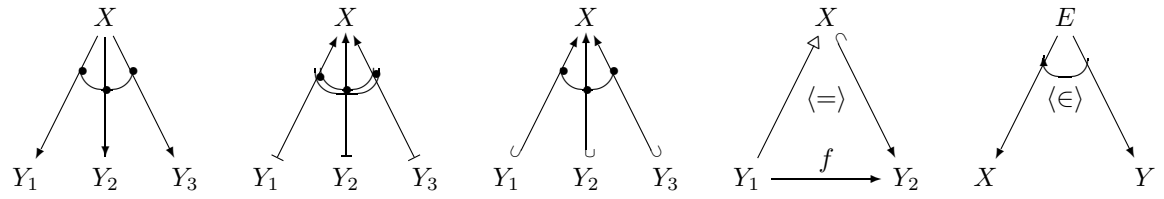


Figure 2: Several simple sketches

3 Data modeling via sketches

Even an overall glance at graphical images currently used in semantic modeling shows an abundance of various kinds of conventional graphical constructs, symbols, labels and markers. Every specialist in semantic modeling and every DB designer uses that kind of semantic schemas [7] which he/she finds more suitable and convenient for one's purposes. This is natural as well as reasonable, however, some problems mentioned in the introduction arise.

It appears that the sketch-based approach provides a proper (with respect to managing heterogeneity) universal framework for data modeling and integration. By suggesting this idea we do not want to force all DB designers to use the same universal specification language; let everyone use that collection of markers he likes. What we actually suggest concerns *what should be marked* in semantic schemas. Indeed, in the majority of semantic schemas in use, particularly, in ER-diagrams which became a kind of the *de facto* standard in the area of conceptual design and modeling, in order to specify the intended semantic meaning of a given node N of a schema, one marks the very node N by a corresponding label. For instance, in the ER-standard to specify a node as a relation, ie, as a set of relationship objects (tuples), one labels it by a diamond. However, if another user perceives the same data objects as entities, he/she labels the corresponding node in his schema by a rectangle. How must be the node N of the integrated schema labeled? The same problem arises if an entity instance in some view is perceived as a set of entities in another view.

One can observe that structural conflicts like above are caused by setting the type of a class via specifying the type of its elements: a relation is a set of tuples, a grouping class is a set of subsets *etc.*

In contrast to thinking in terms of elements, the category theory paradigm of arrow thinking suggests to specify the type of a given class by characterizing (labeling) the corresponding diagram of functions adjoint to the class. Here is a very simple example: the sketch specification of relationships.

Under *a source* we will mean a set, say, X, equipped with a family $\mathcal{F}$ of functions, $f_i\colon X \longrightarrow Y_i,\ i = 1, ..., n.$ Then one has the function $f = [f_1 \dots f_n]\colon X \longrightarrow Y_1 \times \dots \times Y_n$ determined in the standard way: $fx = [f_1x, \dots, f_nx]$. It is easy to see, that f will be injective, ie, X will be 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 very set without imposition of any constraints but constrain instead the corresponding source of outgoing functions to be separating.

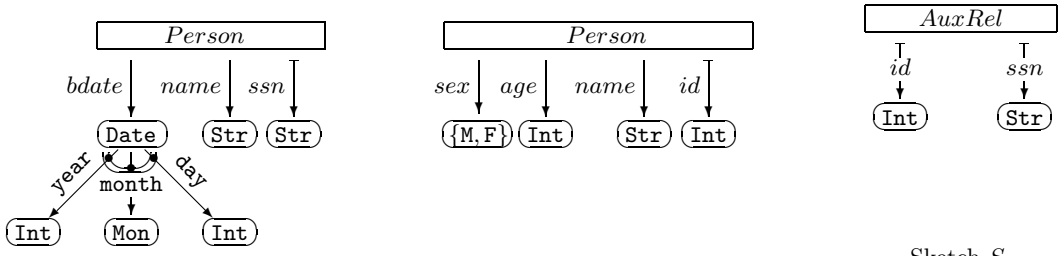
Correspondingly, on the syntax level, to specify a node as a relation one can safely leave the node without any marking but mark instead the corresponding source of outgoing arrows by a label (say, an arc) with the intended semantics of being a separating family of functions. (Actually, this is nothing but a well known idea of designating a key of a relation).

Thus, the leftmost sketch on Fig.2 describes the node X as a ternary relation. We mean that for any extent mapping ε which assigns sets $X^\varepsilon, Y_j^\varepsilon$ to the corresponding nodes, and functions f_j^ε to the corresponding arrows ($j = 1, 2, 3$), if ε satisfies the constraint expressed by the arc marker, ie, if $(f_j^\varepsilon,\ j = 1, 2, 3)$ is a separating family, then X^ε is a relation (up to isomorphism). Further we will explain sketch markers in a semi-formal way without accurate distinguishing syntax and semantics levels. It is hoped that precise formulations can be easily built along lines of the example just described.

A collection of diagram markers is presented in Table 1. For example, according to the table, the following five pictures (sketches) on Fig.2 explain the node X as a relation, disjoint sum (coproduct), union, the image of a given function and a subset of the powerset (of Y) respectively.

Despite their simplicity, the considerations above clearly demonstrate the general possibility of semantic modeling through imposing conditions on arrow diagrams contrary to imposing conditions on nodes as is often done in conventional semantic schemas. Such a machinery makes it possible to avoid structural conflicts mentioned at the beginning of the section but, of course, do not save one from other kinds of conflicts. The general AGO-methodology of managing

[7] Graphical pictures for semantic modeling are often called *diagrams*. However, in category theory this term is already used for a notion being technical but principle for our considerations (not very correctly, *a diagram* over a graph is a collection of nodes and arrows of this graph). So, for the former notion we will use the word 'schema'.



Sketch S_1 Sketch S_2 Sketch S_{aux}

Figure 3: Sketches to be integrated

conflicts in schema integration will be demonstrated in the next section through examination of a simple example.

4 An example of schema integration via sketches and equations

We will consider a very simple example which however makes it possible to demonstrate the essence of the approach and its principal features.

Let two DBs store information about overlapping universes of discourse. The first DB schema consists of a class '*Person*' with the key attribute *SS_No* of the type `String`, and other attributes are *Name*:`String`, *BirthDate*:`Date` where `Date` is the complex type determined by its own attributes [*Day*:`Integer`, *Month*:`Month`, *Year*:`Integer`] with `Month` a user-defined data type consisting of twelve fixed characters.

Let the second DB schema consists of a class also named '*Person*' with attributes *Id*:`Integer`, *Name*:`String`, *age*:`Integer`, *Sex*:`{M,F}` where *Id* is the key and `{M,F}` is a user-defined data type consisting of two fixed characters.

Sketches depicting these schemas are presented on Fig.3 (note monic markers on key attributes).

Oval nodes are predefined *value domains* whose semantics is *a priori* known to the DBMS. Rectangle nodes are *abstract classes* whose extensions should be stored in the DB. [8].

Let us suppose further that there is a federated application which needs data from both DBs, and from the view point of this application '*Person*'s from the first DB are '*Men*', '*Person*'s from the second DB are '*Employee*'s, and, in addition, it is known that the class '*Men*' contains the class of male '*Employee*'s.

[8]For AGO, `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*' is a *name* labeling nodes without imposing any constraints.

To specify this information formally we proceed as follows.

First of all, we extend the second sketch with derived items denoting (a corresponding procedure of computing) the extension of the class of all male persons. Namely, we add to the sketch S_2 the inclusion arrow `m`: `{M}` $\hookrightarrow$ `{M,F}` (see Fig.4), whose existence is derivable from the existence of the set `{M,F}` (`m` is also a marker and a name simultaneously). Then we apply the operation *CoIm* of taking, for a given function, the coimage of a given subset of the codomain to the pair of arrows (*sex*,`m`) , which gives the left square diagram in $\overline{S_2}$ (Fig.4) [9].

The ⟨`CoIm`⟩-marker hung on the diagram just expresses the fact that the corresponding nodes and arrows are obtained by the operation *CoIm*: $(MPrs', mp', !') = CoIm(\mathtt{m}, sex)$.

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$'$,$''$, *, $^\#$ 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.

Now the overlapping between schemas can be presented by the angle $(MEmpl, mem^@, mee)$ (see Fig.5) together with equalities 1,10,11 from the table on the right of Fig.5 (two names standing in the same row denote the equation expressing equality of the corresponding extensions).

Note, however, that the arrow $mem^@$ do not appear in the table and needs special specification.

It is clear that in order to define the arrow $mem^@$ some additional information about correspondence between *ssn*-numbers from S_1 and *id*-numbers from

[9]The sketch notation for graph-based operations is presented in Tabl.2. 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. Thus, Table 2 presents a graph-based query language.

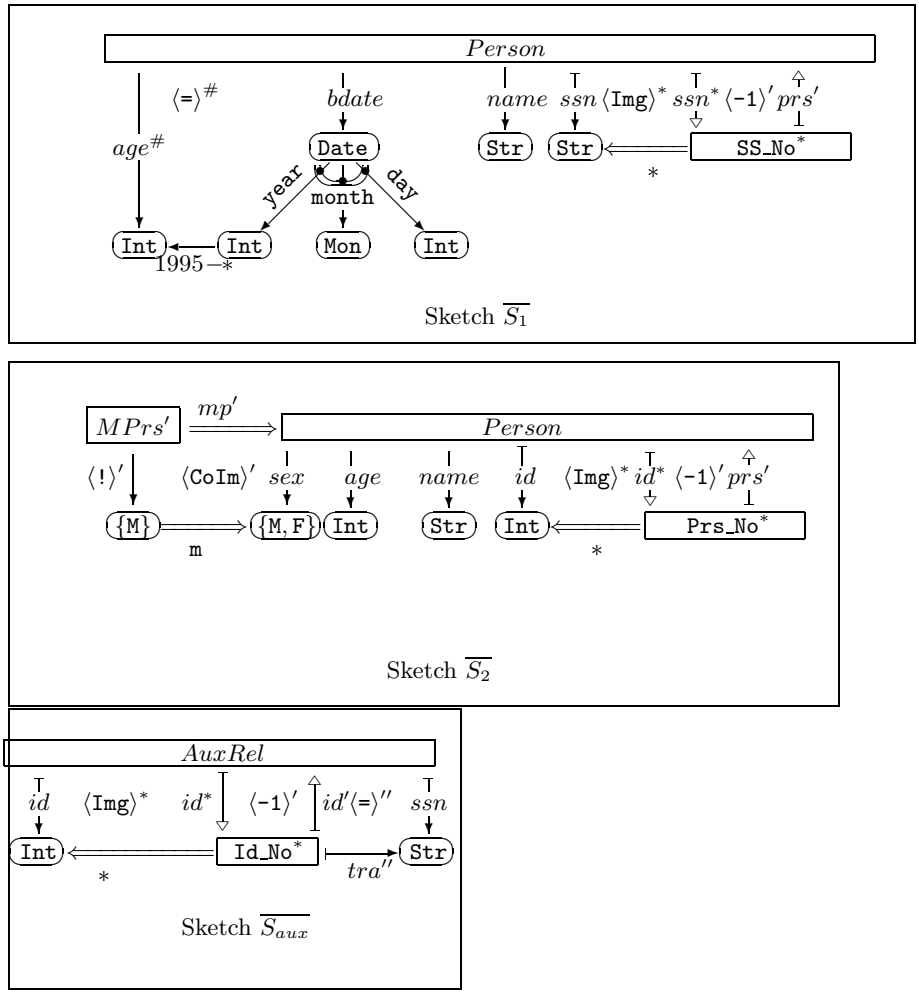
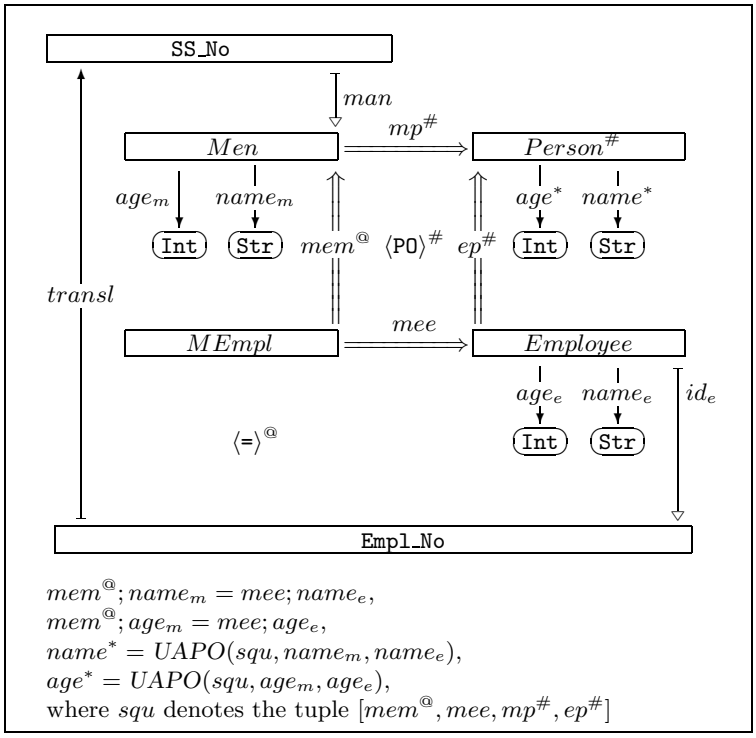


Figure 4: Augmentation of local sketches with derived items



Correspondence Sketch, S_{CI}

No	S_{CI}	$\overline{S_1}$	$\overline{S_2}$	$\overline{S_{aux}}$
1	Men	$Person$		
2	$name_m$	$name$		
3	age_m	$age^{\#}$		
4	man	prs'		
5	$Employee$		$Person$	
6	$name_e$		$name$	
7	age_e		age	
8	id_e		id^*	
9	Empl_No		Prs_No	Id_No
10	$MEmpl$		$MPrs'$	
11	mee		mp'	
12	$transl$			tra''

Correspondence Equations, E_{CI}

Figure 5: Specification of the correspondence information

S_2 is required. We assume that this information is accessible and is stored in a special auxilary table '*AuxRel*' (Fig.3). The arrow $mem^{@}$ can be now defined as the composition of three arrows

$$mem^{@} \stackrel{\text{def}}{=} id_e; transl; man$$

which is depicted in the sketch S_{CI} by the marker $\langle$=$\rangle^{@}$ hung on the corresponding diagram. Note, all component arrows appear as right-hand-side terms in the corresponding equalities of the table E_{CI}. In its own turn, these terms are derived items of local sketches defined by the corresponding sketch operations as presented on Fig.4.

Now it should be hopefully clear how the sketch and the table on Fig.5 were built; simulteneously, 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 obligatorily derived) data as presented on Fig.4. As for the sketch S_{CI}, those markers that are not depictured in the very graph are written in a string notation below the graph. These three linear specifications are an ordinary component of the sketch S_{CI} like other markers specified graphically. Note that two upper equations just provides the possibility of applying the operation *UAPO* (see Tabl.2) for input data specified in two lower equations.

We would like to emphasize that neither the correspondense sketch S_{CI} nor the set of correspondence equalities E_{CI} cannot be built automatically (even if the table *AuxRel* is known). Only the integrating person (DBA) possesses the information on inter-schema correspondence and carries the responsibility of specifying this information, that is, of setting the sketch S_{CI} and equalities E_{CI}. However, the very process of building sketches can be supported by an intelligent graphical editor. In particular, augmentation of sketches with derived items can be assisted by a built-in parsing checker for verifying correctness of hanging operation markers: input items of an operation should be among output items of preceding operations.

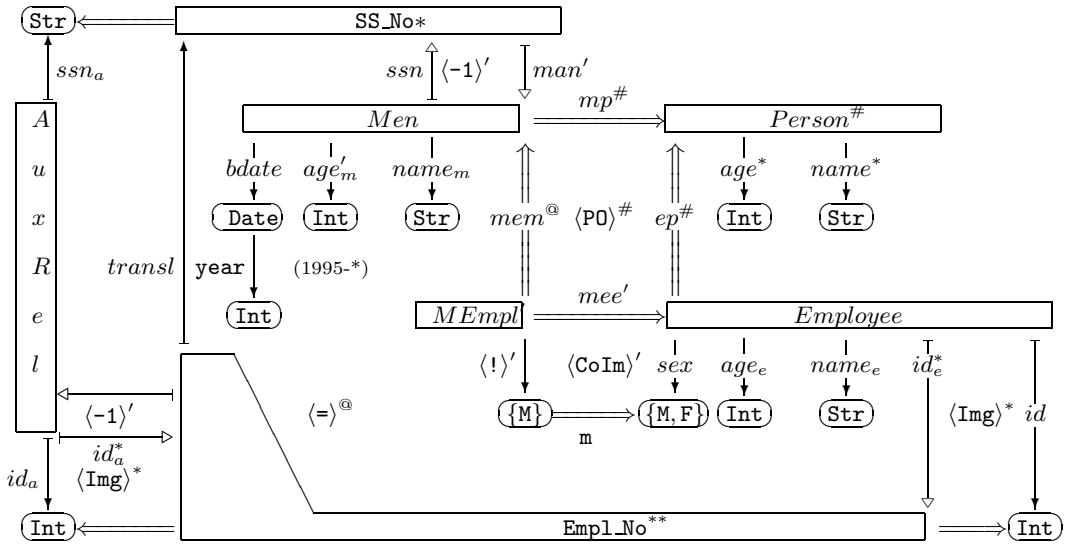
The next step of the integration process can be performed automatedly. It consists in gluing together those nodes and arrows that appear in the correspondence equalities . A resulting glued node/arrow is coloured by green iff at least one of its origins is green. The result of glueing is presented on the Fig.6 .

Black (*ie*, without superscripts) items correspond to the information stored in local DBs, whereas green (indexed by superscripts) items denote derived information, that is, procedures computing it.

Note, the class Empl_No is indexed with two superscripts which mean that this node is obtained by two separate operations simultaneously, namely,

$$\texttt{Empl\_No} = Img(S_2.id) \text{ and } \texttt{Empl\_No} = Img(S_{aux}.id)$$

(the row 9 in the table E_{CI} actually gives two equalities). In fact, this is an *equational* integrity constraint: the set of values in the *id*-column of the table



Full Integrated Sketch (FIS), $\overline{S_I}$

Figure 6: Merging modulo Correspondence Equations

AuxRel should coincide with the set of values in the *id*-column of the relation $S_2.Person$.

We call the result of merging *the full integrated sketch*. The name refers to the fact that this sketch specifies all the basic information stored in local (including auxilary) DBs, and, in addition, some query procedures computing derived information necessary for integration. So, FIS provides federated users with a global consistent schema of data. Of course, for various particular applications a system of views over FIS should be defined.

It is important that the extension of any FIS's item can be computed according to equalities from the set E_{CI} and operation markers appearing in the extended local sketches and in the very FIS. This point provides the principal possibility of converting any query against FIS into a system of queries against local DBs; specific algorithms of such a conversion can be developed depending on local DBMS query languages.

The example just described is very simple but it clearly demonstrates the general AGO-strategy for heterogeneous schema integration. Every particular multibase situation needs elaboration of the appropriate specific signatures of diagram markers and operations similar to those described in Tables 1,2. After that is done, integration can be performed along lines we have described.

Thus, AGO provides a general framework for heterogeneous schema integration whose practical benefits are

- guaranteed universality w.r.to various data models,

- easy adaptability for specific integration tasks,

- clear division of the entire integration procedure into potentially automatic, automated and human-dependent steps,

- orientation on graph-based interfaces (for integration and querying),

- all prerequisites of effectiveness owing to algebraic treatment of the intermediate manipulations.

References

[AH87] S. Abiteboul and R. Hull. IFO: a formal semantic database model. *ACM TODS*, 12(4):525–565, 1987.

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparitive analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[BW90] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall International Series in Computer Science, 1990.

[CD93] Cadish and Z. Diskin. Algebraic graph-oriented approach to view integration. Part I: Specification framework and general strategy. Technical Report 9301, Frame Inform Systems/LDBD, Riga, Latvia, 1993.

[CD94] B. Cadish and Z. Diskin. Algebraic Graph-Oriented = Category Theory Based. Manifesto of categorizing database theory. Technical Report 9406, Frame Inform Systems, Riga,Latvia, 1994. (On ftp: .../MANIFEST/mnfst*.ps).

[DH84] 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.

[Dis92] Z. Diskin. A mathematical framework for comparing expressive powers of different data model.

In *Methods of Database Design, MDBD'92*, Baltic Int.Conference, pages 28–33, Riga, Latvia, 1992.

[DKM+93] 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.

[HK87] R. Hull and R. King. Semantic database modeling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[LNE89] 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.

[Mot87] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE TOSE*, 13(7):785–798, 1987.

[Nav92] S. Navathe. The next ten years of modeling, methodologies and tools. In *Entity Relationship modeling, ER'92*, number 645 in Springer LNCS'645, 1992.

[Shi81] D. Shipman. The functional data model and the data language DAPLEX. *ACM TODS*, 6(1):140–173, 1981.

[SL90] A. Sneth and C. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 1990.

[SP91] S. Spaccapietra and C. Parent. Conflicts and correspondence assertions in interoperable databases. *ACM SIGMOD Record*, 20(4):49–54, 1991.

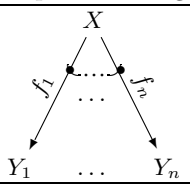
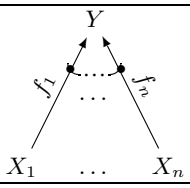
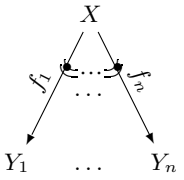
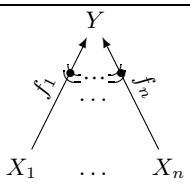
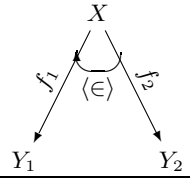
[SPD92a] S. Spaccapietra, C. Parent, and Y. Dupont. Model-independent assertions for integration of heterogeneous schemas. *Very Large Databases Journal*, 1(1), 1992.

[SPD92b] S. Spaccapietra, C. Parent, and Y. Dupont. View integration: a step forward in solving structural conflicts. *IEEE Transactions on KDE*, 1992.

[SST92] 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.

[WHW90] S. Widjojo, R. Hull, and D. Wale. A specificational approach to merging persistent object bases. In *4th Int.Workshop on Persistent Object Systems*, 1990.

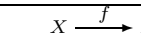
[YAD+92] K. Yetongnon, M. Andersson, Y. Dupont, S. Spaccapietra, H-J. Schek, M. Scholl, and M. Tresch. The FEMUS experience. In *Semantics of interoperable database systems, IFIP DS-5*, 1992.

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 $X \overset{f}{\subset\!\!\rightarrow} Y$	$X \subset Y$ 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†	$X \overset{f}{\longrightarrow\!\!\triangleright} Y$	$Y = f(X)$
Inversion	$X \underset{g}{\overset{f}{\langle \texttt{-1} \rangle}} Y$	$(\forall y \in Y) f(g(y)) = y$
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 $Y_1 \longmapsto$ **Powerset**Y_2 .
Pull-Back of ISA-Arrows	$\begin{array}{ccc} X_1 & \Longrightarrow & Y \\ \Uparrow & \langle \texttt{PB} \rangle & \Uparrow \\ Z & \Longrightarrow & X_2 \end{array}$	$Z = X_1 \cap X_2$
Push-Out of ISA-Arrows	$\begin{array}{ccc} X_1 & \Longrightarrow & Y \\ \Uparrow & \langle \texttt{PO} \rangle & \Uparrow \\ Z & \Longrightarrow & X_2 \end{array}$	$Y = (X_1 \setminus Z) \cup (X_2 \setminus Z) \cup Z$

† The constructions tagged with † are nothing than trivial cases of previous markers.

Table 1: A collection of diagram constraints

Table 2. **Sketch Operations**

Name	Designation in texts	Designation in diagrams	Arity Shape: Input sketch	Arity Shape: Output sketch	Denotational Semantics
Identity	*Id*	⟨id⟩			$(\forall x \in X)\ f(x) = x$
Gluing	*Glue*	⟨!⟩			$Y = \{*\};\ \ (\forall x \in X)\ f(x) = *$
Image	*Im*	⟨Img⟩			$I = \{f(x) :\ x \in X\}$ $(\forall x \in X)\ f'(x) = f(x)$
Composition	;	⟨=⟩			$(\forall x \in X)\ f(x) = g_2(g_1(x))$
Graph	*Graph*	⟨Gra⟩			$G = \{(x, fx) :\ x \in X\}$ p, q are projections
Equivalizer	*Eqvr*	⟨Eq⟩			$E = \{x \in X : fx = gx\}$
Inversion	$()^{-1}$	⟨-1⟩			$(\forall x \in X) g(f(x)) = x$
Push-out	*PO*	⟨PO⟩			$S = \{(a,1) : a \in (A \setminus C)\} \cup$ $\{(b,2) : b \in (B \setminus C)\} \cup$ $\{(c,3) : c \in C\}$
Universal Arrow of PO	*UAPO*	⟨!⟩			$(\forall s \in S)$ $u(s) = \begin{cases} i(a) & \text{if } s = (a,1), \\ j(b) & \text{if } s = (b,2), \\ i(c) = j(c) & \text{if } s = (c,3) \end{cases}$
Pull-back	*PB*	⟨PB⟩			$R = A \cap B$
Coimage	*CoIm*	⟨CoIm⟩			$A = \{x \in X :\ fx \in B\}$ $f' =$ restriction of f on A