Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

Chapter 3

WHAT VS. HOW OF VISUAL MODELING: THE ARROW LOGIC OF GRAPHIC NOTATIONS

Zinovy Diskin
F.I.S. Group, Latvia
zdiskin@acm.org
diskin@fis.lv

Boris Kadish
ZAKAZ.COM, Inc.
bkadish@zakaz.com
cadish@fis.lv

Frank Piessens
Dept. of Computer Science
K.U.Leuven, Belgium
frank@cs.kuleuven.ac.be

Abstract The goal of the paper is to explicate some universal logic underlying various notational systems used in visual modeling. The idea is to treat the notational diversity as the diversity of visualizations of the same basic specificational format. It is argued that the task can be well approached in the arrow-diagram logic framework where specifications are directed graphs carrying a structure of diagram predicates and operations.

1. INTRODUCTION

People like drawing pictures in order to explain something to others or to themselves. Concise graphical images – *visual models* – presenting various aspects of the universe of discourse are a very natural (for humans) means of communication and comprehension, and proved their practical helpfulness in a wide range of design activities: from thinking out how to knock a shack (where a drawing is useful yet optional) to design of high-rise buildings of business/software systems (impossible without visual models). The history of graphic notations invented in various scientific and engineering disciplines is rich and instructive but it is software/business engineering where for the last years one can observe a Babylon diversity of visual modeling languages and methods: ER-diagrams and a lot of their dialects, OOA&D-schemas in a million of versions and, at last, the recent UML which itself comprises a host of various notations. There is a plenty of literature on the subject and a huge amount of papers were written to advance a few basic ideas in one or another direction or polish them up to one or another level of details.

In this respect, what we would like to say clearly from the very beginning is that we see the present paper as *not* yet another one towards greater generality or a bit more well-defined semantics. Our goal is much more ambitious: to clarify the basic semantic foundations of visual modeling (VM) on a whole, and present an integrated framework

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

where many problems of VM can be approached consistently, or, as our experience shows, even be eliminated at all by a proper formulation. Our concern is about *what* we actually do by visual models rather than inventing any particular, even "the best one", VM-notation. However, a clearly defined *what* induces certain guidelines for *how* and we will point to some basic principles of building VM-notations, which our analysis of semantic foundations suggests.[1] We call the corresponding graphic format *sketch* following mathematical category theory (CT) where (somewhat different) sketches were invented for specifying mathematical structures.[2]

Let S be a specification in some language $\mathcal{L}$ whose details are not essential, particularly, $\mathcal{L}$ can be some VM-notation and S is a graphic schema (we prefer to call graphic specifications schemas rather than diagrams because the term *diagram* has special technical meaning in the sketch language). If one is well satisfied with informal intuitive meaning of S, we have almost nothing to say about benefits of sketches. However, as soon as one is interested in some abstract, intuition-free, description of the meaning of S, the sketch format immediately appears. Indeed, a lesson one can learn from modern mathematics is that formal semantics of any notation should be searched in the world of abstract homogeneous sets and functions, and sketches are nothing but a graphical language for specifying this world and its constructs. Moreover, a surprising result proved in CT states that *any* formal construction can be expressed by a sketch and, thus, as soon as S has a meaning described formally, it can be expressed by some sketch S_0 as well. As for S, its syntax can be arbitrary (and even not graphical at all) but, in principle, we can consider it as a syntactic visualization sugar over the basic sketch structure S_0 behind S. Generally speaking, this visualization superstructure is independent of S_0 but the meaning of S would be much more transparent if some parallelism between formal logical specification S_0 and its visual appearance S holds. This at once leads to a useful principle of graphic notation design: in a reasonable language $\mathcal{L}$, any schema S appears as $V(S_0)$ with S_0 a logical sketch specification behind S and V a mapping of specificational constructs into their visualizations.

Despite the absolute expressiveness of the sketch language, its vocabulary of basic terms is extremely brief. Roughly, a *sketch* is a directed graph in which some diagrams (graph's fragments closed in a certain technical sense) are marked with labels taken from a predefined *signature*; these labels (markers) are predicate symbols and marked diagrams are nothing but predicate declarations. So, a sketch specification consists of only three kinds of items: (i) nodes, (ii) arrows, (iii) marked diagrams. Formally, it can be presented by a tuple $[G, P_1(D_1), \ldots, P_n(D_n)]$ with G a graph, P_i diagram predicate symbols and $P_i(D_i)$ predicate declarations for diagrams D_i in G (to be read as "diagram D_i of sets and functions possesses the property P_i"). In addition, some of predicate symbols may denote diagram operations (similar to presenting, say, addition of integers by a ternary predicate); thus, (iv) an attribute of being *basic* or *derived* should be defined for all items of a sketch. The four constructs we mentioned exhaust the basic sketch vocabulary.

In the sketch framework the task of designing a VM-notation in some applied field (semantic data modeling, behavioral modeling, business modeling, X-modeling with X an arbitrary domain of discrete nature) is reduced to the following two subtasks:

(i) design of sufficiently expressive and convenient signature of diagram predicates to build *logical* sketches S_0,

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

(ii) design of appropriate way of external presentation of sketch specifications, that is, design of sketch *visualization mechanism* V to build graphic specifications, *visual* sketches, $S = V(S_0)$.

This clear formulation is in sharp contrast with the current practice of VM-notation design often based on shaky and *ad hoc* logical foundations (if any).

The task (i) needs formal explication of domain specific constructs in the sketch language: it is a typical applied mathematics setting similar to, for example, application of calculus in mechanical or electric engineering. The absolute expressive power of sketches does guarantee feasability of this task (at least, in principle, but of course within the very limits of intuitive domain formalizability). However, for real practical applications of some notation $\mathcal{L}$, more important than the expressive power of $\mathcal{L}$ are naturality of $\mathcal{L}$-specifications for the domain, their practical effectiveness and convenience, and such an immaterial (yet integral and we believe very important for vitality of complex intellectual systems including notations) property as elegance. In this respect, the visualization part of notation design is of extreme importance but it is far beyond formal logic and the sketch format as such, actually, it is a non-trivial problem of general cognitive nature. Nevertheless, the rigor and clear specification pattern provided by sketches allows to approach this problem by mathematical methods, in particular, to consider visualization V as a morphism from some structure of specificational primitives into a similar structure of visual primitives.[3]

Well, the sketch language is expressive but we emphasize that, as any other logical machinery, it does not solve the two major problems (i),(ii) above and, given some particular discipline of VM, the virtual possibility of expressing its modeling constructs in the sketch language has nothing to do with effective usage of sketches in the field. However, the clear formulation of graphic notation problems in the sketch framework, its brevity and the semantic guidelines it suggests are themselves helpful and organize the design activity on reasonable (and mathematically justified) foundations. At any rate, they allow to focus on resolving actual problems having real semantic meaning rather than fighting with pseudo-problems generated exclusively by using an unsuitable *ad hoc* language and existing only within that language. A typical example is the infamous problem of integrating ER and OO that has been widely discussed in the literature on conceptual data modeling but at once disappears as soon as one approaches it in a proper semantic framework [DK].

As always with applications of abstract mathematics to engineering problems, applying sketches to system/business modeling is an art rather than science and much depends on heuristic issues that cannot be formally deduced or predicted. So, discussing practical benefits of adopting sketches for visual modeling in a particular field $\mathcal{F}$ we can speak only about expectations rather than assert firmly, and only real "case studies" of notations invented in $\mathcal{F}$ and problems they should help to manage can confirm or refute our "sketch thesis". In this respect our current experience is promising: in domains where we applied sketches they appeared as a precise, and practically useful, logical refinement and generalization of the existing notation rather than an external imposition upon it. In particular, in semantic data modeling sketches can be seen as a far reaching yet natural generalization of functional data schemas, ER-diagrams and OOA&D-schemas ([DK]), in meta-specification modeling they appear as essential generalization of schema grids developed by the Italian school [BBS93] (see [Dis98a]

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

for the sketch treatment of meta-modeling) and we expect that in process modeling sketches will be a natural development of interaction diagrams [Abr94].

The rest of the paper consists of bits of additional explanations, few examples and few comments to points mentioned above, all are necessarily very brief because of space limitations. In section 2.1 we explain the distinction between logical specification as such and its visual presentation. We believe that this issue is of principle importance for VM but often is not properly understood. Then we briefly discuss current approaches to building semantic foundations of VM and their principle shortcomings, and explain basics of the sketch solution we suggest, particularly, its benefits for unification goals (section 3.). Section 4. is a very brief outline of the semantic data specifications in the sketch framework. It is only one example of refining a VM-practice with sketches but hopefully it gives a general notion how the machinery works. In particular, in 4.3 we will compare notational mechnisms of our sketches and two particular kinds of UML-diagrams.

2. FORMAL SEMANTICS FOR GRAPHIC NOTATIONS

To be really useful as business and software specifications, and as a means of communication between many people of diverse backgrounds involved in design of complex systems, VM-diagrams (VMDs) should have clear and unambiguous semantic meaning. However, semantic meanings of VMDs' items live in the world of abstract concepts, they cannot be "pointed out by finger" and must be described in some special language, necessarily symbolic to be able to refer to abstract concepts. Certainly, since the description should be unambiguous the language for specifying semantics must be extremely precise, in ideal, formal or ready to be formalized.

So, the academic problem of formal semantics for a given VM-notation attains a real practical value, and many attempts were made to build precise semantic foundations under ER-modeling, OO-modeling and, in particular, now we can observe an explosion of semantic research for the UML-modeling. All these attempts can be roughly divided into the two large groups briefly discussed below in sections 2.2 and 2.3 but to make the discussion substantial we first need some methodological preparation in section 2.1.

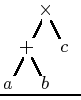
2.1 GRAPH-BASED VS. STRING-BASED LOGICS

It is crucial for understanding the purposes of the present paper to recognize the distinction between string-based and graph-based logics. The point is that any specification – as it is presented to its reader – is actually a *visual presentation* of a certain underlying *logical specification* as such. In general, there are possible graphical visualizations of string-based logical specifications and, conversely, linear string visualizations of graph-based specifications.

Let us consider, for example, an expression "$(a+b)\times c$" where a, b, c denote elements of some predefined domain and symbols $+, \times$ denote binary operations, say, *plus* and *prod*, over the domain. The expression denotes the result of applying *prod*-operation to the pair (r, c) where r denotes the result of applying *plus*-operation to the pair (a, b). So, the logical specification hidden in the visual expression $(a + b) \times c$ is a parsing tree which can be presented in the bracket notation by the formula $prod(plus(a, b), c)$.

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

This specification can be extracted from the visualization above if the parser knows that operations *plus* and *prod* have (arity) shapes as shown in the Version1-column of the table below (bullets • denote place-holders for which elements of the domain are to be substituted). If one takes other visualizations of the shapes, one will get other visualizations of the same logical specification (see the table for examples).

	Version1	Version2	Version3
Visual shapes of operations	$\bullet + \bullet$, $\bullet \times \bullet$	$\bullet + \bullet$, $\prod(\bullet, \bullet)$	
Visual presentation of specification $prod(plus(a,b),c)$	$(a+b) \times c$	$\prod(a+b, c)$	

In shapes of operations above one can distinguish the logical part – a set of place-holders called the *arity* of operation, and the visualization part – the mutual arrangement of place-holders and the symbol (marker) of the operation. To distinguish different place-holders in an arity shape it is reasonable to label them by natural numbers and then an arity appears as a string of place-holders. Hence, the logic we consider is string-based. As for the visualization mechanism, it can be linear (*eg*, columns 1,2 of the table) or two-dimensional (column 3). Correspondingly, the specification above will be visualized by a linear string or a two-dimensional image as is shown in the bottom row of the table. In the Version3-case the image is a graph but it does not mean that the logic itself is graph-based. More complex yet real examples of the same notational phenomenon are the so called conceptual graphs [MMS93] and graphical interfaces to relational database schemas employed in many design tools. Conversely, graph-based logic specifications can be presented in a linear plain form: after all, a graph is an ordinary two-sorted mathematical structure well specified by formulas.

So, in considering graphic notational systems one should carefully distinguish between specification and visualization, and graph-based logics should be carefully distinguished from graphical interfaces to string-based logics. Any logic with string-like arities is a string-based logic, no matter how graphical its visualization looks.

2.2 FOREIGN FORMAL SEMANTICS: VISUAL MODELS AS STRING-BASED ELEMENT-ORIENTED THEORIES

With the approach in the title, visual models are considered as theories in some string-based logical formalism built along the lines the ordinary many-sorted predicate calculus ($\mathcal{PC}$) is built. Examples (just to mention few) are $\mathcal{PC}$-like formalization of OO built in [KLW95], of ER in [GH91] and of UML fragments in [LB98, WJS94]. However, $\mathcal{PC}$-like formalizations of graphic notations used in conceptual modeling fail to capture the two key ideas behind VM:

(i) in syntax, stating *graph-based logic* as the main apparatus,

(ii) in semantics, viewing the world as a system of objects and object classes inter-related by associations/references between them.

Indeed, syntax of $\mathcal{PC}$ is based on sets (strings) rather than graphs though, as it was shown above, these string specifications can be graphically visualized. As for semantics, $\mathcal{PC}$ enforces one to view the world in terms of elements and op-

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

erations/relations over these elements, that is, constitutes an *elementwise* approach (where, for example, tuples are primary while relations are derivative – they are sets of tuples). In contrast, the OO-modeling necessarily leads to viewing the world as a collection of homogeneous variable sets (classes) and functions (references) between them while internal structure of elements (objects) is derivative: for example, first a set R is somehow declared to be a relation over domains $D_1, \ldots D_n$ and only then elements of R can be considered as $D_1 \times \ldots \times D_n$-tuples. This OO-view is somewhat opposed to the elementwise one and, thus, the $\mathcal{PC}$-approach does give precise semantics but for something different than modern VM-notations.

2.3 NATIVE NAIVE SEMANTICS: VISUAL MODELS AS GRAPH-BASED OO-DESCRIPTIONS

In this approach, semantic meaning of VM-diagrams has been explicated in some direct way by considering VMDs immediately as graphic specifications with their intended OO-interpretations. These attempts are very different technically but share the naive approach to the problem when formal semantics for a graphic notation has been built in an *ad hoc* way based on common sense and intuition. We call these approaches naive since actually the issue is highly non-trivial and hardly can be properly developed from scratch. Indeed, it was in a focus of CT research in 1960s-70s and stating a proper basic framework required serious efforts of the mathematical community. The corresponding foundations are now well known in CT under the name of *topos theory* and, in fact, the most advanced studies in searching VMD-semantics resulted in rediscovery of particular topos-theoretic concepts in a naive way. However, as a rule, approaches of this second kind provide "semantics" that is just a different yet again highly informal re-description of intuitive meaning (the UML Semantics [RJB99] is an example).

2.4 NATIVE FORMAL SEMANTICS: VISUAL MODELS AS SKETCHES

The only way to describe something abstract in an unambiguous way is to describe this thing in some also abstract yet commonly understandable basic terms. For a long time this problem was in a focus of mathematical studies and to the present day two frameworks are established. One is the classical set-theoretical framework based on the key notions of *set* and its *elements* related with the fundamental *membership* predicate. This is the ground of $\mathcal{PC}$-semantics and elementwise thinking referred above. The second framework is more recent, it was developed in CT and is based on two other key concepts: of *object* (*eg*, set) and of *morphism* (*eg*, mapping). Syntactically, specifications in this second framework take the form of directed graphs consisting of nodes (objects) and arrows (morphisms), and the entire approach is often called *arrow logic* and even *arrow thinking* since its applications assume a special ability of specifying any universe of discourse by arrows.

It should be stressed that the difference between the two approaches is not only in that specifications in the former are strings while in the latter they are graphs. Much more important is the substantial difference in the intended semantic interpretations. Normally, items of string specifications are interpreted by elements of some predefined

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

domain (or several domains), and by constructs built from elements. In contrast, items of the arrow specifications are normally interpreted by set-like (for nodes) and mapping-like (for arrows) objects of some predefined universe of domains and mappings between them. As for some internal "elementwise" structure of these objects, it is characterized externally via adjoint arrow diagrams: all that one wants to say about an object one has to say via arrows related to this object (in section 4. we will show a few simple examples how it can be done).[4] As it was formulated by one of the founders of the arrow thinking Bill Lawvere, "to objectify means to mappify". In contrast, in the first framework, to "objectify" something means to point out elements of this thing.[5]

A fundamental result proved in CT is that expressive powers of the two languages are equal: any set-element specification can be converted into an object-arrow one expressing the same semantics and vice versa. In addition, this common expressive power is *absolute*: by a basic dogma of modern mathematics, the very notion of formalizability means the possibility to be described via sets and elements, hence, via objects and arrows as well. So, for *any* universe of discourse, as soon as one needs to build its formal model, one can well consider the universe as a system of nodes and arrows, which, in addition, carries a certain structure of arrow diagram predicates; that is, as a category theorist would say, to consider the universe as a *topos*. Then visual models of the universe are nothing but finite specifications of the topos (in general, infinite) and, as such, can be *naturally* considered as sketches we described in introduction: indeed, sketches are nothing but finite presentations of toposes.

3. TOPOSES AND SKETCHES: SEMANTIC DIVERSITY IN THE SAME SPECIFICATIONAL FORMAT

3.1 SKETCHES AND THEIR INTERPRETATIONS

A topos is an abstract mathematical structure whose items (nodes, arrows, markers) can be manipulated in an abstract algebraic way. In concrete applications, one deals with concrete toposes consisting of, for example,

(i) sets (data types) and functions (procedures), in *functional programming*;

(ii) variable sets (object classes) and variable functions (references), in *semantic data modeling* / *OO analysis and design*;

(iii) propositions and proofs, in *logic/logic programming*;

(iv) interfaces and processes, in *process modeling*.

In other words, when one specifies a universe in a discipline above by a sketch, nodes and arrows of the sketch can be interpreted, in function of context, as pointed. One can also say that sketch nodes and arrows are formal parameters while their interpretations above are actual parameters.

A special and important interpretation is when nodes and arrows are considered as

(v) specifications and mappings between them (*meta-modeling*).

It was shown in [Dis98a] that in this way the basic notions of view and refinement of specification can be modeled and, thus, the arrow language turns out to be very natural for meta-modeling.

So, in different applications we have different toposes yet they are instances of the same abstract construct. Then, all the semantic diversity above can be managed within the same mathematical and notational framework. For example, a relation between

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

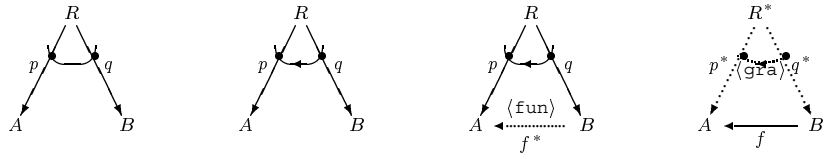


Figure 3.1 Relations via arrows

two objects A and B of whatever sort they are (for example, A, B can be data types, or object classes, or process interfaces, or data schemas) is specified in the same way by an arrow span shown on schema (a) Fig. 3.1. In this figure, R is an object similar to A, B, which is to be thought of as consisting of pairs (links, relationships) of items of A, B, and p, q are projection morphisms (references, processes, schema mappings) to be thought of as extracting the first and the second components of the pairs. In addition, a special predicate *Rel* should be declared for the triple (R, p, q) to ensure that R-items can be indeed thought of as pairs of A-items and B-items. A natural way to express such a declaration $Rel(R, p, q)$ syntactically is to mark the arrow diagram (p, q) by some label, say, an arc, as shown on schema, in fact, a sketch, (a). The description is generalized for n-ary relations in a straightforward way: consider arrow n-spans $(R, p_1, ..., p_n)$ over domains $A_1, ..., A_n$.

Specifically, the property of being a functional relation, say, from B to A, can be expressed by another predicate declared for the same diagram; syntactically, it can be presented as, *eg*, shown on sketch (b), Fig. 3.1. In this case there is a mapping $f\colon\ B \to A$ derived from the relation. In fact, one has a *diagram operation* producing an arrow from a span marked with the functional relation label, see schema (c), where the marker $\langle \mathtt{fun} \rangle$ denotes the operation in question and the dotted body of the f^*-arrow denotes that it is derived (by the operation). Conversely, from a given arrow (reference, process, schema mapping) $f\colon\ B \to A$, one can derive its graphic (R^*, p^*, q^*). Formally this can be expressed as application of the operation $\langle \mathtt{gra} \rangle$ to the arrow *f* as shown on (d) where all derived items are marked with the *-superscript. So, in the arrow framework relations can be specified, and manipulated, in an abstract way without considering their elements but via their arrow interfaces, (p, q) in the example.

3.2 SKETCHES VS. HETEROGENEITY OF VISUAL MODELS

In the previous section we discussed possibilities of the unifying sketch format across different fields where VM is applied. However, even within the same field usually there are many different VM-methodologies resulting in different notational systems. For example, in conceptual data modeling there are sets of ER-based and of OO-based notations, and many "this-vendor-tool-based" ones. With the current trend to cooperative/federal information systems, one necessarily encounters severe problems of integrating specifications describing (overlapping fragments of) the same universe but in different notational systems. In fact, the famous UML recently adopted as a standard in OO analysis and design, is nothing but an attempt to solve this sort of

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

problems "once and for ever". No doubts that the UML is a significant achievement towards unification but, from the logical view point, it is just another (very bulky) notational system rather than a framework to manage the heterogeneity problem.

In the sketch framework the heterogeneity problem can be approached as follows. First of all we note that actually any sketch is a Π-sketch, where Π denotes some predefined signature of diagram predicates and operations (markers) that can be declared in the sketch. Very different signatures are possible, and each of them determines its own notational subsystem within the same sketch language.

Now, given some field $\mathcal{F}$ of applying VM, let $\mathcal{M}$ be a particular VM-methodology used in the field. One can arrange $\mathcal{M}$-vocabulary of specificational constructs into a signature, $\Pi_{\mathcal{M}}$, so that $\mathcal{M}$-specifications become convertible into $\Pi_{\mathcal{M}}$-sketches. Thinking semantically, this is always possible since a given $\mathcal{M}$-specification and its $\Pi_{\mathcal{M}}$-sketch counterpart are nothing but different presentation of the same semantic universe, that is, of the same topos (of $\mathcal{F}$-sort). Moreover, by adjusting visualizations of $\Pi_{\mathcal{M}}$-predicates one can make visual presentations of $\Pi_{\mathcal{M}}$-sketches close to $\mathcal{M}$-schemas as they are seen externally. In this way the *diversity* of VM-models (used in $\mathcal{F}$) can be transformed into the *variety* of sketches in different signatures. Indeed, sketches in different signatures are nevertheless sketches, and they can be uniformly compared and integrated via relating/integrating their signatures. Though the latter task is far from being trivial, it is precisely formulated and can be approached by methods developed in CT.[6] Anyway, in many interesting particular cases the signature integration is easy.

The methodology of the sketch approach can be illustrated by the following analogy. Let us consider applications of sketches to semantic data modeling. The place of sketches in the heterogeneous space of semantic meta-models is comparable with that of the modern positional notations for numbers in the general space of such notations (we will call them numeral systems) including also zeroless positional systems (*eg*, Babylonian, Mayan) and a huge diversity of non-positional *ad hoc* systems (Egyptian, Ionian, Roman *etc*). The analogy we mean is presented in the table below.

Specification paradigm	Data to be specified	Language		Minimal language
		Predefined base	Specification	
Positional numeral systems	Finite cardinalities	Base of numeral system, k	Positional k-number	Binary numbers
Sketches	Collections of sets and functions	Signature, Π	Π-sketch	Original categorical sketches

In the context of this analogy, many conventional notations used for semantic data modeling are similar to *ad hoc* non-positional numeral systems like, *eg*, Roman. The question about expressive power of sketches is analogous to the question of whether a positional system can emulate an arbitrary numeral system. The positive answer is evident to everybody but it would not be so if one considers the question within the pure syntactical frame: thinking syntactically, it is not so obvious how to translate Roman numbers into decimal. The question above is easy because of our inherited habit to think about numbers semantically: any Roman number is a presentation of some finite cardinality, and a decimal number can express the latter as well.

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

The situation with sketches is similar: thinking semantically, any data schema is a specification of a system of sets and functions, and the latter can be expressed by a sketch as well. Of course, a specialist in semantic modeling who has the habit to think of conceptual schemas in pure relational terms will have doubts whether an arbitrary complex logical formula can be expressed by a diagram predicate. The translation is indeed far from being evident but nowadays can be found in any textbook on categorical logic (*eg*, [BW90]).

Well, sketches do provide a principle possibility to manage heterogeneity of visual models in a consistent way by translating any model into a sketch. However, for practical applications the crucial question is how easy and natural such a translation could be, and whether the visual appearance of the model and its sketch counterpart could be a good match. The latter is important since a given visual notation often accumulates some useful experience and VM-techniques developed in the field; at any rate, it is habitual for the community and useful notational habits surely should be kept. As for semantic data modeling, it was shown in [Dis98b] how one can put conventional notational constructs into the sketch pattern by considering them as either special visualizations of diagram markers or as special abbreviations. In sections 4.3, 4.4 below we will present an example of such a case study by sketching two familiar UML-constructs.

4. EXAMPLE OF SKETCHING A VM-DISCIPLINE: SEMANTIC DATA MODELING VIA SKETCHES

The main idea underlying the sketch approach to data modeling is to consider object classes as plain sets consisting of internally unstructured (homogeneous) elements whereas all the information about their type is moved into certain arrow (reference) structures adjoint to classes.

4.1 COMPLEX TYPES VIA ARROW DIAGRAMS

The arrow way of specifying basic complex types (tuple type, set type, variant type) was presented in [Dis98b]. Here we will reproduce the tuple type description to make the paper selfcontained on the first reading level.

Instead of saying that a set X consists of n-tuples over a list of domains $D_1, \ldots, D_n$, one can equivalently say that there is a *separating* family of functions, $f_i\colon\ X \longrightarrow D_i (i = 1, \ldots, n)$, that is, a family satisfying the following condition:

(Sep) for any $x, x' \in X,\ x \neq x'$ implies $f_i(x) \neq f_i(x')$ for some i

Indeed, in such a case the tuple-function $f = \langle f_1 \ldots f_n \rangle$ into the Cartesian product of domains,

$$f = \langle f_1 \ldots f_n \rangle\colon\ X \longrightarrow D_1 \times \cdots \times D_n, \quad fx \stackrel{\text{def}}{=} \langle f_1 x, \ldots, f_n x \rangle$$

is injective so that elements of X can be considered as unique names for tuples from a certain subset of $D_1 \times \ldots \times D_n$, namely, the image of f. In fact, elements of X can be identified with these tuples so that X is a relation up to isomorphism. In the classical ER-terminology, if the domains D_i's are entity sets then f_i's are *roles* and any $x \in X$

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

is a relationship between entities $f_1(x), \ldots, f_n(x)$. Since, conversely, for any relation the family of its projection functions is separating, the very notions of a tuple set and separating source (of functions) are equivalent.

Correspondingly, on the syntax level, to specify a node as a relation one may leave the node without any marking but label instead the corresponding source of outgoing arrows by some marker (say, an arc) denoting the (Sep)-constraint.

4.2 INTERPRETATION OF ARROWS

The key point in semantics of sketches is how to interpret arrows. In the former considerations we interpreted arrows by ordinary functions, *ie*, totally defined single-valued functions. This is the standard category theory setting. In contrast, in semantic modeling it is convenient (and common) to use optional and multivalued attributes/references, and so other interpretations of arrows arise: by partially defined functions (*p-functions*) and by multivalued functions (*m-functions*); of course, interpretations by partially defined and multivalued, *pm-functions*, are also possible.

Note, m-functions and p-functions are not ordinary functions subjected to some special constraints. Just the opposite, a single-valued function is a special m-function $f : A \longrightarrow\!\!\!\!\blacktriangleright\, B$ when for any $a \in A$ the set $f(a) \subset B$ consists of a single element. Similarly, a totally defined function is a special p-function $f : A \circ\!\!\longrightarrow B$ whose domain $D_f \subset A$ coincides with the entire source set A. So, to manage optional multi-valued attributes and references in the sketch framework we assume that

(i) all arrows are by default interpreted by pm-functions,

(ii) there is an arrow predicate (marker) of being a single-valued function,

(iii) there is an arrow predicate (marker) of being a totally defined function.

It is convenient to visualize constraintless arrows (without markers) by $\circ\!\!\longrightarrow\!\!\!\!\blacktriangleright$ whereas $\longrightarrow\!\!\!\!\blacktriangleright$ and $\circ\!\!\longrightarrow$ are denotations of arrows on which markers are hung: the ordinary tail is the marker of being totally defined and the ordinary head is the marker of being single-valued. Of course, superposition of these markers is also legitimate and it is natural to visualize it by the arrow $\longrightarrow$. Thus, visualization of predicate superposition equals superposition of visualizations: here we have a simple instance of applying a useful general principle that reasonable graphic notation should follow. (Actually, it gives rise to a consistent mathematical framework for building graphic notation we mentioned in introduction).

4.3 SIMPLE EXAMPLES: SKETCHES VS. UML-DIAGRAMS

We will consider two very simple examples to give some, actually very rough, notion of conceptual modeling with sketches.

The UML-diagram D_1 on Fig. 3.2(i) describes a very simple universe: each *Villa*-object has one or more owners from the class *Person* and a *Person*-object may own none or only one *Villa*; *Ownership* is a corresponding association class as they are understood in the UML [BJR99].

The corresponding sketch specification is presented on the right. Two arrows out of the *Ownership*-node with the arc-marker hung on them (section 4.1) show that *Ownership*-objects can be considered as pairs (v, p) with $v \in [\![\, Villa \,]\!]$ and $p \in [\![\, Person \,]\!]$, here $[\![\ ]\!]$ denotes semantic mapping assigning sets to nodes and functions

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

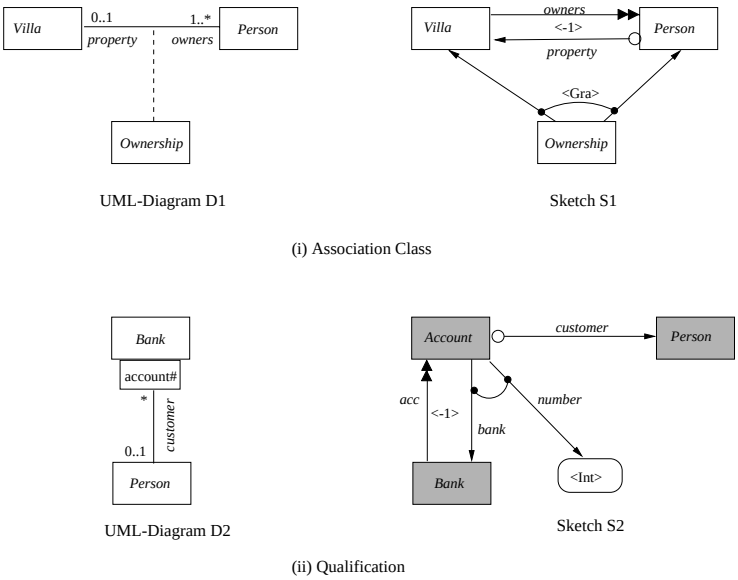


Figure 3.2 Sketches vs. UML: two examples

to arrows. Tails and heads of the horizontal arrows (section 4.2) declare the same constraints as the left and right superscripts over the association edge on the UML-diagram. Marker $\langle$`-1`$\rangle$ hung on the pair of horizontal arrows denotes the predicate of being mutually inverse functions. Marker $\langle$`Gra`$\rangle$ means that $[\![\ Ownership\ ]\!]$ is the graph of (each of) these functions, that is, if $(v, p) \in [\![\ Ownership\ ]\!]$ then $v = [\![\ property\ ]\!](p)$ and $p \in [\![\ owners\ ]\!](v)$.

Another example is presented on Fig. 3.2(ii). The UML-diagram D_2 models the so called *qualification* construct when two coupled nodes are attached to the same end of an association edge: this means that in order to select an object on the other end of the association one should point out a value of the qualifier. In the example, *account#* (smaller rectangle) is a qualifier and thus, for a given *Bank*-object b, while $b.customer$ is the set of b-customers, $b[55555].customer$ is the single *Person*-object having account #55555 at the bank b.

Sketch specification of the same data semantics is presented on the right. A peculiarity of sketch S_2 is that it has two kinds of nodes. Rectangles denote object classes (in the database context, extension of these nodes should be stored and it is suggestive to fill-in them with dots) while stadions denote value domains (computable rather than storable).

Semantics of this latter kind of nodes is *a priori* known to the DBMS and, thus, for the sketch approach, $\langle$`Int`$\rangle$ is a *marker* hung on the corresponding node and denoting the corresponding constraint (predicate). Indeed, if a node is marked by $\langle$`Int`$\rangle$ its intended semantics is constrained to be the predefined set of integers. At the same time, *Person* and *Bank* are *names* labeling nodes without imposing any constraints.

Actually, specifications we have just considered capture only a very poor projection of real semantic phenomena of association and qualification. The point is that semantic

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

meaning of major conceptual modeling constructs (object identity, IsA- and IsPartOf-relationships, various association and qualification relationships) can be precisely explicated only in the framework of *variable* sets semantics for sketches ([DK], see also [Dis98b] for a shorter presentation). In this framework, sketch nodes and arrows are interpreted by sets and functions changing in time while markers denote invariant (constant in time) properties of variable set-and-function diagrams. This interpretation enriches conceptual modeling with a new – dynamic, or evolutionary, – dimension and in the much more rich dynamic world (well known in CT under the name of *topos of variable sets*) there are several kinds of dynamic separation predicates and several kinds of dynamic "simply functions", correspondingly, there are several kinds of dynamically different associations and qualifications. For example, a few different *Ownership*-associations are possible, but when one interprets sketch items by static sets and functions, all this variety is degenerated into a purely structural, and much more poor, picture which we have specified by sketch S_1. In Appendix the basic idea is very briefly motivated and outlined.

4.4 VISUALIZATION ASPECTS

Having two given notational samples for comparison, the question which one is "more right" is incorrect: any notation with unambiguously specified semantics can be used. However, the question of which notation is more clear and transparent w.r.t. its intended semantic meaning is quite reasonable.

Compare, for example, UML-diagram D_1 and sketch S_1 on Fig. 3.2(i), which express the same semantics. On a whole, the sketch presents a more detailed specification – all functions involved are shown explicitly – while the UML-diagram can be considered as a special abbreviation. In fact, in D_1 we have two different abbreviations. One is the presentation of two mutually inverse functions by one undirected edge each of whose ends carries the corresponding name and multiplicity constraint. It is a reasonable abbreviation, it makes the graphical image more compact and multiplicity expressions are as good as special arrow heads and tails, or even better w.r.t. mnemonic efforts. Note, however, that when one considers compositions of many references, undirected visualization can lead to mistakes. As for abbreviating the arrow span out of node *Ownership* in the sketch S_1 by an edge going into another edge in D_1, it is a purely syntactical trick hardly clarifying semantic meaning of the association class *Ownership*. In addition, such a way of presenting binary associations does not work for multiple (greater than 2) arity associations. In contrast, in the sketch language arbitrary n-ary association are presented by n-arrow spans in a uniform way.

Diagram D_2 on Fig. 3.2(ii) is even a more powerful abbreviation: four nodes and four arrows of the actual semantic picture (specified by sketch S_2 in detail) are compressed in a graphical image with only three nodes and one edge. However, the distance between visual schema D_2 and its semantic meaning ("congruently" specified by sketch S_2) is so large and meandering that diagram D_2 hardly can be considered as presenting a good visualization mechanism.

Of course, the issue we are discussing is of complex cognitive nature and such culture-dependent points as notational habits, preferences, notions of elegance can play significant role; analysis of such things goes far beyond the sketch formalism as such. Nevertheless, we believe that clear logical structure of sketch specifications

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

and the presence of well-defined semantics for them make the sketch format a proper foundation for building a really good graphic notational system upon it.

Finally, concerning visualization of sketches, it must be emphasized that visualization on a computer display is much clearer than visualization on paper. Colours can be used to distinguish between basic and derived items in the sketch. Dynamic highlighting of marked diagrams is very useful if one has to deal with complex sketches with many markers. In that case, it can be difficult to visualize on exactly which diagram each marker is hung. On a computer display, a satisfactory solution is to highlight a marked diagram in response to a click on the marker.

5. CONCLUSION: IS IT OF REAL PRACTICAL INTEREST?

The platform we have outlined provides VM with consistent and mathematically justified semantic foundations. However, as it was quite justly noted in [KR98], the user of visual models does not need to use (or even see) the underlying framework. In this respect we would like to stress specially that the sketch-topos view has also quite immediate practical consequences for VM even in its current, mainly *ad hoc*, state. The major of them is that VM-diagrams should be directed graphs while in the majority of VM-notations undirected edges are employed. Another one is that many markers used in graphical schemas could, and should, be considered as denoting diagram predicates and operations in, generally, a variable sets-and-functions world. At least, clear separation of predicate/operation markers from those performing another, for example, meta-specificational or purely syntactic-sugar function is quite necessary. Also, explicit notational distinction of the fundamental difference between basic and derivable items is very important for VM.

On a whole, the topos-sketch view we suggest gives rise to a whole program of refining the VM-vocabulary, making it precise and consistent, and unified. In the topos framework for semantics, *any particular VM-notation appears as a particular visualization of the same common specificational format* – the format of sketches. Besides this unifying function, an essential advantage of the sketch format is the extreme brevity of its conceptual vocabulary. Nevertheless, as it was discussed in the paper, the sketch language is absolutely expressive and possesses a great flexibility. So, sketches enjoy a nice (in fact, unique) amalgamation of graphical evidence, rigor and expressiveness.

Acknowledgments

We are indebted to Ilya Beylin (Chalmers University) for many disputes on the sketch notation, which always were stimulating and substantial. We are also grateful to our colleagues at F.I.S, George Sheinkman and Vitaly Dzhitenov, for encouragement-from-practitioners helpful in both technical and moral aspects. Special thanks go to Joseph Goguen for a few brief yet substantial discussions by correspondence, which clarified some important aspects of the problem we approach. Finally, most of all we are indebted to Haim Kilov with whom the ideas presented in the paper were repeatedly discussed, considered and re-considered from very different viewpoints; in particular, these discussions made the structure and presentation of our material in the paper much clearer.

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

Appendix: Formalizing mysterious object identity

Real world sets are changeable and consist of changeable objects. For example, the set of persons of some community at a time moment t, $[\![Person]\!]^t$, and conceptually the same set at another moment u, $[\![Person]\!]^u$, are physically different. Speaking accurately, elements of the sets $[\![Person]\!]^t$, $[\![Person]\!]^u$ are *states* of persons at the moments t, u rather than persons themselves and hence, in general, the sets $[\![Person]\!]^t$ and $[\![Person]\!]^u$ are even disjoint. However, these sets are not mutually independent: they are inter-related by identifying different elements, say, $p' \in [\![Person]\!]^t$ and $p'' \in [\![Person]\!]^u$, as different states of the same person, say, p. In this case we write $p' = p(t)$ and $p'' = p(u)$.

What is p? In the ER data model a suitable concept is that of entity, in the OO-paradigm p would be called an object identity (o-id). However, in these and other data models the notion is not formally explicated and remains intuitive. In contrast, we take the constructive ontology view point that "to be" means to be observed, and so the identity is the *observable* identity. That is, if there are some verified reasons (a constructive proof as a mathematician would say) to consider p' and p'' to be (the two different states of) the same entity, this should be declared in an explicit form. Such declarations, irrespectively to ways how the information could be obtained, can be expressed by a family of binary inter-state relations

$$^t[\![Person]\!]^u \subset [\![Person]\!]^t \times [\![Person]\!]^u,\ t < u \text{ are time moments,}$$

so that $p' \in [\![Person]\!]^t$ and $p'' \in [\![Person]\!]^u$ have to be considered as different states of the same entity if and only if $(p', p'') \in {}^t[\![Person]\!]^u$.

Of course, object identity is a construct for building a logical model of a universe rather than writing its chronicle: time moments s, t, u should be treated generally as indexes of real world states, and their precedence is logical rather than physical. Correspondingly, the logical time is branching: it is ordered partially rather than linearly.

Definition. Let $\mathcal{T} = (T, \leq)$ be a logical time and $\mathcal{O}$ be a class of *observable states* of objects. A $(\mathcal{O}, \mathcal{T})$-*variable set* C is a family of sets $[\![C]\!]^t \subset \mathcal{O},\ t \in T$, taken together with a family of binary *interstate relations* $^t[\![C]\!]^u \subset [\![C]\!]^t \times [\![C]\!]^u,\ t \leq u$, such that the following *transitivity* condition holds for any $s \leq t \leq u \in T$:

$$(\text{Trans})_{s \leq t \leq u} \qquad {}^s[\![C]\!]^t \ \bowtie\ {}^t[\![C]\!]^u \ =\ {}^s[\![C]\!]^u,$$

here $\bowtie$ denotes the operation of relational composition.

When the context $(\mathcal{O}, \mathcal{T})$ is fixed, we call $(\mathcal{O}, \mathcal{T})$-variable sets simply variable sets, or *varsets*. C is the name of the varset in question, $[\![C]\!]^t, t \in \mathcal{T}$ are *states of* C and elements of $[\![C]\!]^t$ are *states of* C*-objects*.

The objects themselves are nothing else than sets of their states indexed by time moments. In particular, if objects do not merge nor split and so all the interstate relations are of one-to-one type, an object c is represented by a trajectory in the space of observable states:

$$c = [c(t_1), c(t_2), c(t_3) \ldots],$$

where $[c(t_i), c(t_{i+1})] \in {}^{t_i}[\![C]\!]^{t_{i+1}}$ and $t = t_1$ is the first moment when some element from $[\![C]\!]^t$ was identified as (state of) c.

Thus, in the variable sets framework an object *is* a trajectory as above and the meta-physical absolute o-id is replaced by a constructive notion of observable trajectory in the space of observable states.

Definition. Let C be an object class, that is, a varset. C-objects are called:

- *Static*, when for any $t \leq u$, $[\![C]\!]^t \cap [\![C]\!]^u$ is not empty and $^t[\![C]\!]^u$ is its diagonal, then object trajectories are glued into points; examples are values like integers or strings.
- *Proper*, when all $^t[\![C]\!]^u$ are of one-one type; examples are entities in the conventional sense like *person*s or *villa*s.

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

- *Legacyless* or *snapshots*, when ${}^t[\![C]\!]^u = \emptyset$ for all $t < u$ and so each object trajectory is defined only for a single time moment – such objects have neither a history nor future; examples are relationships in the sense of the ER-model like *ownership*s between *person*s and *villa*s.

Examplifying legacyless objects by conventional relationships needs explanation. What we have said is that there is a certain view on *ownership*s, say, *R-view*, in which pairs $(p', v') \in [\![Person]\!]^t \times [\![Villa]\!]^t$, $t \in T$ are temporally discrete. This means that the question of whether the same pair (p', v') in two different (but related) states, $t < u \in T$,

$$(p', v') \in [\![Person]\!]^t \times [\![Villa]\!]^t \text{ and } (p', v') \in [\![Person]\!]^u \times [\![Villa]\!]^u,$$

represents the same *ownership* or another *ownership* of the same objects is not stated, or considered incorrect (an example of different *ownership*s between the same p and v is when p sold his villa v but then purchased it again in another terms).

Of course, there is another view on the ownership, *E-view*, when identity of *ownership*s is traced through time. Then the question of identifying two pairs of the same objects is legitimate, moreover, mandatory, and we consider the ownership as an entity association rather than a legacyless relationship.

Remark. In both cases, when a varset C is legacyless and when it is a value domain, o-id trajectories of C-objects are points, but note an essential difference. In the first case trajectories are defined only for a single moment, in fact, the very notion of trajectory is degenerated. In the second case we have trajectories but they are of special kind: they are reduced to points.

Finally, it should be stressed that predicates of being a snapshot, or a value, or a proper object, are determined by observer's (user's) needs or possibilities rather than intrinsic properties of objects. More details of the varset taxonomy can be found in [DK, section 4].

The formal framework provided by ideas described above is still not sufficient for proper semantic data specifications. Indeed, the real world is full of ontological inter-dependencies between objects: objects may form associations, may be parts of other objects, may be views to other objects and so on. Variable set semantics explicates the notion of object identity but what is to do with dependencies between objects like those listed above?

Fortunately, the answer is an almost immediate consequence of the varset idea. Indeed, because of identity of C-objects is explicated by interstate relations ${}^t[\![C]\!]^u$, $t \leq u$, one can say that ontology of C-objects is given by C-interstate relations. Then, a natural way to explicate that C-objects somehow ontologically depend on objects of other classes $D_1, D_2, \ldots, D_n$ is to state that interstate relations for C depend on those for D_i, that is, ${}^t[\![C]\!]^u$ can be somehow derived from ${}^t[\![D_i]\!]^u$, $i = 1, \ldots, n$, for each pair of moments $t \leq u$. For example, given object classes *Employee* and *Person*, the meaning of declaration "any *employee* is a *person*" (usually it is expressed by declaring *Employee* to be an IsA-subclass of *Person*) is that object identity of *employee* coincides with that of *person*s. Formally this can be expressed as follows: (i) there is an arrow (reference) $f\colon\ Employee \to Person$, which is semantically interpreted by a variable one-one function

$$[\![f]\!]^t\colon\ [\![Employee]\!]^t \to [\![Person]\!]^t,$$

and (ii) interstate relations for *Employee* are directly determined by those for *Person*:

$$(e', e'') \in {}^t[\![Employee]\!]^u \text{ iff, } \textbf{by definition}, (e'.[\![f]\!]^t, e''.[\![f]\!]^u) \in {}^t[\![Person]\!]^u$$

for any $t \leq u \in T$, $e' \in [\![Employee]\!]^t$, $e'' \in [\![Employee]\!]^u$. In other words, object identity of the class *Employee* is derived from that of *Person* via the function f.

What was just described for the IsA-relation between objects is a general idea. It is shown in [DK] (see [Dis98b] for a shorter presentation) that in a similar way semantic constructs of association, aggregation, qualification, IsPartOf relationships can be explained and formalized. So, what is intuitively perceived as ontological difference can be formally explicated dynamically in the varset framework.

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

Notes

1. Of course, VM is a complex human activity and should be considered in the corresponding contexts, including, in particular, highly non-trivial issues of cognitive and social nature far beyond formal logic and technical semantics for VM-diagrams (cf. [Gog96]). The latter are not only specifications but also political documents (especially, if we consider high-level visual models) and thus some fuzziness and vagueness are unavoidable, and even useful [Gog98]. Nevertheless, we believe that clear technical semantics picture is helpful anyway, in particular, it prevents some technical problems with serious social consequences. On the other hand, as was noted by one of the editors of this book, some room for manoeuvres so important for political documents can be also achieved by using quite precise yet incomplete specifications admitting a few possible realizations. At any rate, in the present paper we focus on the specificational half of visual models leaving social aspects aside.

2. References to CT-literature should be an important component of the present paper. However, we are not aware of CT-books suitable for software engineers or business modeling people. And indeed, a good book of such kind could only be the result of certain experience of applying CT to practical problems and teaching CT to the corresponding audience. Unfortunately, the former is still very rare, and the latter does not exist at all, at least, in a systematic form. Moreover, as soon as one starts to apply CT to real problems in business/software engineering, (s)he at once realizes that the arrow machinery – as it was developed in mathematical CT – itself needs serious adaptation and development towards generalizations helpful in practice, some consideration on this issue can be found on first pages of [Dis97]. So, we restrict ourselves by a single reference [BW90]: it is a good source of required CT-results, and is very well written from the view point of theoretical computer science though not very suitable for a non-initiated reader.

3. Close ideas were developed by Goguen and his group in their studies of building reasonable graphical interfaces [Gog97].

4. Interpretations of arrow specifications when nodes are interpreted by elements (object instances rather than classes) are also possible, then arrows are binary relationships of partial order or instances of logical consequence. In this framework, propositional logic can be sketched.

5. Arrow diagrams adjoint to a node C can be considered as interfaces to the class C and so the idea of the approach can be phrased as that objects show themselves only through arrow diagram interfaces. Such a setting appears to be a precise realization of the basic OO principle of encapsulation, the more so that arrows can be treated as formal counterparts of references, attributes and methods. Thus, conceptually, objects in the OO sense and objects in CT have much in common.

6. Quite briefly, there is some set of basic diagram operations such that semantic meaning of any diagram predicate can be defined in terms of these operations and equations; as a result, any signature of diagram predicates can be considered as an essentially algebraic theory over some common set of basic operations. This makes it possible (at least, in principle) to relate and compare sketches in different signatures in some unified way.

References

[Abr94] S. Abramsky. Interaction categories and communicating sequential processes. In A.W. Roscoe, editor, *A Classical Mind: Essays in honour of C.A.R.Hoare*, pages 1–15. Prentice Hall Int., 1994.

[BBS93] C. Batini, G. Battista, and G. Santucci. Structuring primitives for a dictionary of entity relationship data schemas. *IEEE Trans.Soft.Engineering*, 19(4):344–365, 1993.

[BJR99] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language user guide*. Addison-Wesley, 1999.

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

[Dis97] Z. Diskin. Generalized sketches as an algebraic graph-based framework for semantic modeling and database design. Technical Report M9701, University of Latvia, 1997.

[Dis98a] Z. Diskin. The arrow logic of meta-specifications: a formalized graph-based framework for structuring schema repositories. In B. Rumpe H. Kilov and I. Simmonds, editors, *Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications)*, TUM-I9820, Technische Universitaet Muenchen, 1998.

[Dis98b] Z. Diskin. The arrow logic of visual modeling and taming heterogeneiuty of semantic models. In H. Kilov and B. Rumpe, editors, *Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications)*, TUM-I9813, Technische Universitaet Muenchen, 1998.

Published in: "Behavioral specifications of businesses and systems", Eds.Kilov et al, Kluwer Publ.Co, 1999

[DK] 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*

[GH91] M. Gogolla and U. Hohenstein. Towards a semantic view of an extended entity-relationship model. *ACM Trans.Database Systems*, 16(3):369–416, 1991.

[Gog96] J. Goguen. Formality and informality in requirement engineering. In *Requirement engineering, 4th Int. Conference*, pages 102–108. IEEE Computer Society, 1996. (keynote address).

[Gog97] J. Goguen. Semiotic morphisms. Technical report, University of California at San Diego, 1997. TR-CS97-553.

[Gog98] J. Goguen. Personal letter, 1998.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal ACM*, 42(4):741–843, 1995.

[KR98] H. Kilov and B. Rumpe. Overview of the Second ECOOP Workshop on Precise behavioral semantics (with an Emphasis on OO business specifications). In *The Europian Conference on Object-Oriented Programming, ECOOP'98*, LNCS 1543. Springer, 1998.

[LB98] K. Lano and J. Bicarregui. Formalising the UML in structured temporal theories. In H. Kilov and B. Rumpe, editors, *Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications)*, TUM-I9813, Technische Universitaet Muenchen, 1998.

[MMS93] G.W. Mineau, B. Moulin, and J.F. Sowa, editors. *Conceptual graphs for knowledge representation*. Number 699 in LNAI. Springer, 1993.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[WJS94] R. Wieringa, W. de Jonge, and P. Spruit. Roles and dynamic subclasses: a modal logic approach. In *European Conference on Object-Oriented Programming, ECOOP'94*, Springer LNCS, 1994.

About the Authors

Zinovy Diskin got MSc in mechanical engineering from Bryansk Inst. of Transport Engineering (Russia) in 1981 and worked in the wagon-building industry for 8 years. In 1992 he got PhD in mathematics from the Omsk State University (Russia) and in 1994 Dr. Math. from the University of Latvia. Current research interests are in graphical specification languages and notations, semantic data modeling and database semantics, abstract algebraic logic and categorical logic. In 1992-1997 he headed Lab. for database design at F.I.S. Currently, he works as a consultant in conceptual and business modeling.

Boris Kadish graduated from the Riga Polytechnic Institute in 1980. In 1986 he defended his Dissertation on database design at the State SRI for Problems of Computer Technique and Informatics in Moscow (Russia). In 1992 the University of Latvia conferred the Dr. of Computer Science degree on him. Since 1991 he has been a scientific advisor of the group supported by Grant No.45 from the Science Council of Latvia "Methods and Integration Tools for Heterogeneous Database Design". In 1992 he was the initiator and the program chair of the international conference "Methods of Database Design". He has a numerous year experience in creating large database systems in Latvia and in former USSR countries. Since July 1998 he is living in California. Since September 1998 he is a co-owner and the president of Californian company ZAKAZ.COM, Inc.

Frank Piessens obtained a Master of Engineering in Computer Science degree from the Katholieke Universiteit Leuven in 1991. He obtained a PhD in Computer Science from the same university in 1996. He is currently a Postdoctoral Fellow of the Belgian National Fund for Scientific Research. His research has focused on applications of category theory to computer science.