On Mathematical Foundations for Business Modeling[1]

Zinovy Diskin[2]
Frame Inform Systems, Ltd, Latvia.
zdiskin@acm.org

1. Introduction

Business modeling (BM) is probably as old as business itself yet only recently it has started to recognize itself as a special engineering discipline that could be taught, facilitated with computer and performed in an industrial environment. Obviously, transformation into an engineering discipline needs elaboration of methodologies, reuse patterns, convenient notations, tools and "how to do" books, all that constitutes technology as opposed to heuristics. However, the long history of engineering shows that a well stated engineering discipline needs well stated mathematical foundations, that is, a consistent framework of mathematical models for engineering models of the domain. In contrast to technology concentrated on "how", mathematics, first of all, explicates "what it is" and "what is to be done", and on this base then contributes to the "how" component by offering convenient notation and effective apparatus (often already existing and polished up to a great extent long before and without any relation to the engineering domain in question).

Of course, mathematical refinement becomes really useful only on a certain stage of development of an engineering discipline, the latter should achieve a certain maturity to be ready for applying mathematics. And BM, as it is presented in the book [7], in a wide context and on a good abstract level far beyond particular notational systems and methodologies, has evidently matured and is ready for inculcating mathematics yet explicit mathematical account of its foundations is still an open problem.

The present sketch-paper aims at sketch-answers to the following three questions:

- What is a business domain, mathematically?
- What is a business model of business domain, mathematically?
- What is the mathematical machinery suitable for building and manipulating business models?

2. What a business model specifies, "engineeringly"

The goal of a particular project in BM is to specify the universe of discourse in abstract and precise terms, and present it in a way easily comprehended by humans. A quite natural specification paradigm, that became widely spread lately, is to consider the universe consisting of objects mutually related and interacting in certain ways. Normally, it's reasonable to merge similar objects into classes and, thus, the universe appears as a collection of object classes.

Of course, these classes are not independent, they are interrelated by various relationships usually called *semantic*. Examples are

- *aggregation*, when we consider, for example, each *Family*-object as an aggregation of a *Man*-object in the role of husband and a *Woman*-object in the role of wife (in italic are the names of classes involved),

[1] Published in Proc. *TOOLS'37*: 37th Int. Conference on Technology of Object-Oriented Languages and Systems (Sydney, Australia), IEEE Computer Society Press, 2000; pp.182-187

[2] Supported by Grants 93.315 and 96.0316 from the Latvia Council of Science

- *IsA*-relationship, when we consider each *Man*-object as a *Person*-object having some additional special properties,
- *composition*, when we consider each *Car*-object as composed from a *Body*-object, *Engine*-object, and four *Wheel*-objects,

and a few other familiar BM-constructs. A common tendency in the BM-literature is to consider semantic relationships as relations over universes of classes: IsA- is binary and aggregation and composition are (n+1)-ary with n=2,3,… . In other words, the situations referred above are specified by the following predicate declarations:

(FAM**)** **Aggregation** (*Family, Woman, Man),* **IsA** (*Woman, Person),* **IsA** (*Man, Person)*;
(CAR) **Composition** (*Car, Body, Engine, Wheel)*;
where predicate (relation) names are typed in bold.

Thus, the base of a typical business model is some universe of object classes together with a set of relations declared over it. Over this base, the second component of the business model — transactions changing the state of the system — are specified. In this paper we concentrate on the structural base leaving mathematical aspects of the transaction superstructure for another presentation.

3. What a business model specifies, mathematically

To explicate the mathematical essence of the construction above we need, first of all, a precise formal model for the notion of an object class (understood extensionally, that is, as a set of instances). It is not a trivial question since a natural and immediate idea to treat classes just as sets of objects turns out to be a too rough approximation. Indeed, the mathematical sets are static and are given once and forever while real world collections are changeable and consist of changeable objects. On the other hand, a natural variation of the set idea — an object class is a *variable* set (*varset*)— provides a really flexible framework where all the semantic structural constructs can be explained and adequately specified formally [3, 4].

3.1. Object classes as variable sets

Briefly, the statement that an object class, say, *Person* is a varset, means the following. Given a time moment t, we can consider the collection of persons in question as a set $[\![Person]\!]^t$ and for another time moment u we will have another set $[\![Person]\!]^u$. 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 P. In this case we write $P' = P^t$ and $P'' = P^u$. So, if there are some verified reasons (a constructive proof as a mathematician would say) to consider P' and P'' to be (two different states of) the same entity, this should be declared in an explicit form. The totality of all such declarations, irrespectively to ways how the information could be obtained, can be expressed by a binary inter-state relation

$${}^t[\![SamePerson]\!]^u \text{ or else } {}^t[\![Person]\!]^u \subset [\![Person]\!]^t \times [\![Person]\!]^u,$$

with $t < u$ are time moment. So, $P' \in [\![Person]\!]^t$ and $P'' \in [\![Person]\!]^u$ have to be considered as different states of the same entity, $P'=P^t$ and $P''=P^u$, if and only if, $(P',P'') \in {}^t[\![Person]\!]^u$.

Thus, a class is not a set but a chain of sets interconnected by binary relations between them, which trace the identity of the class' objects. In this varset framework, semantic relationships between classes are specified in the following way.

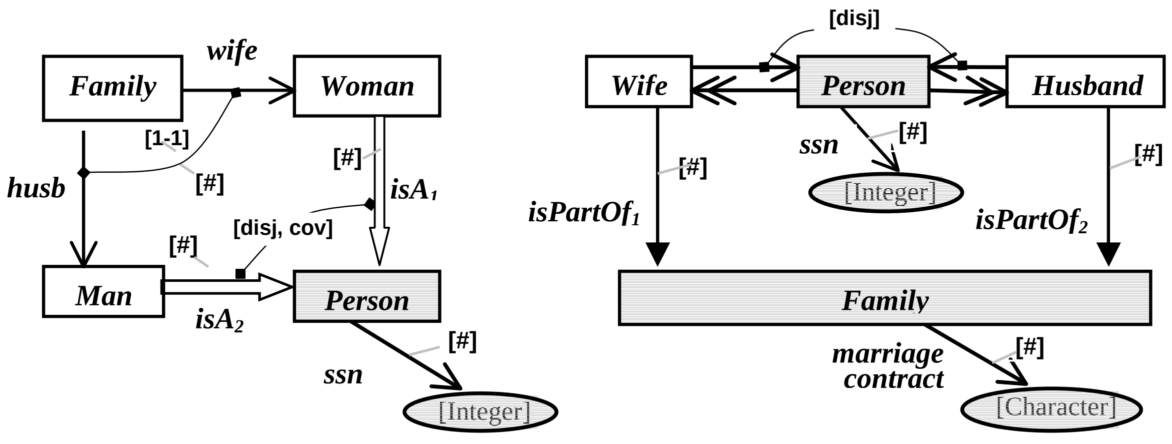
3.2. Semantic relationships between classes via arrows

The first key idea is to enter onto the stage arrows -- mappings between classes (varsets). Consider, for example, the arrow presentation of a "Family domain" on Fig.1a (cf. (FAM) description in section 2). Ordinary arrows denote single-valued mappings between classes;

block arrows are special mappings, inclusions, which send an object into itself but considered from a different view point — as a member of another class. Note, expressions isA_1, isA_2 are just arrow names like *husb* and *wife* with no formal semantics attributed to them. To express the latter we need other means.

A part of domain's semantics is captured by predicate declarations (shown in square brackets) for arrow diagrams. For example, conjunction of predicates [disj] and [cov] is declared for the diagram of arrows (isA_1, isA_2) taken together with their source and target nodes. It means that images of sets *Man* and *Woman* are disjoint and cover the set *Person* at any time moment, formally, $\llbracket Woman\rrbracket^t \cap \llbracket Man\rrbracket^t = \varnothing$ and $\llbracket Woman\rrbracket^t \cup \llbracket Man\rrbracket^t = \llbracket Person\rrbracket^t$ and for any t. The marker [1-1] is hung on the span diagram in the left top corner and means that the mapping sending each *Family*-object F to the pair ($F.wife$, $F.husb$) provides a one-one correspondence between them. Note, we assume that our universe of *Family* objects is considered for some (long) period of time and, hence, neither *wife* nor *husb* references taken separately do not provide 1-1 identification of *Family*-objects.

However, the declarations above have nothing to do with ontological semantic relationships declared in (FAM) in section 2. Indeed, the [1-1]-property of the arrow diagram (*wife*, *husb*) provides a one-one correspondence between *Family* objects F's and pairs ($F.wife$, $F.husb$) but it does not mean in anyway that *Family*-objects **are** such pairs. To express the latter, we apply the second key idea of the approach: to model semantic relationships between classes by (derivability) relationships between their interstate relations.



a) Class *Family* as an aggregation b) Class *Family* as a composition

Figure 1. Semantic relationships via arrows

For example, considering the class *Family* as an aggregation of *husband* and *wife* amounts to the following. First of all, it means that a family is totally identified by its husband and wife and we talk about "the family of Mary and John" rather than "the Smith family" or "the family created by the marriage contract #A11". Formally, this means that a *Family*-object is identified by the values of its *husb* and *wife* references: for any time moments $t \le u$ and objects $F' \in \llbracket Family\rrbracket^t$, $F'' \in \llbracket Family\rrbracket^u$,

$$(F', F'') \in {}^t\llbracket SameFamily\rrbracket^u \text{ iff, by definition, } (F'.\llbracket wife\rrbracket^t, F''.\llbracket wife\rrbracket^u) \in {}^t\llbracket SameWoman\rrbracket^u \text{ and } (F'.\llbracket husb\rrbracket^t, F''.\llbracket husb\rrbracket^u) \in {}^t\llbracket SameMan\rrbracket^u$$

In other words, object identity of the class *Family* (explicated by the totality of its interstate relations) is derived from those of the classes *Woman* and *Man* (via the mappings *wife* and *husb*). Syntactically, this is expressed by hanging a special marker # on the corresponding span diagram (Fig.1a).

Similarly, when we say that "a man is a person" we mean that object identity of *Man*-objects coincides with that of *Person*-objects: for any $t < u$, $M' \in \llbracket Man\rrbracket^t$, $M'' \in \llbracket Man\rrbracket^u$,

$(M', M'') \in {}^{t}[\![SameMan]\!]^{u}$ iff, by definition, $(M'.[\![isA_2]\!]^{t}, M''.[\![isA_2]\!]^{u}) \in {}^{t}[\![SamePerson]\!]^{u}$,
and thus object identity of the class *Man* is derived from that of *Person* via the function isA_2. Syntactically, this is again expressed by hanging the marker # on the arrow-identifier. In addition, nodes for classes with derived object identity are transparent while nodes for basic classes whose object identity is independent of other classes (the class *Person* in our case) are filled.

The situation with the composition construct is a little bit more complicated. The statement that the class *Wheel* is a component of the composite class *Car* means that *Wheel*-objects loss their individuality and appear only as "wheels of certain cars": we say "the left front wheel of the car #A11". Similarly, it is possible to consider a family as a composition of wife and husband and then we say "the wife/husband of the family with marriage contract number #A11" (note the crucial difference from our first view where we talked about "the family of Mary and John"). Thus, in the composition view we have a basic class *Family* and two dependent classes *Wife* and *Husband* whose object identity (interstate relations) are derived via arrows $isPartOf_{1,2}$ (Fig.1b) as follows:

$(W', W'') \in {}^{t}[\![SameWife]\!]^{u}$ iff $(W'.[\![isPartOf_1]\!]^{t}, W''.[\![isPartOf_1]\!]^{u}) \in {}^{t}[\![SameFamily]\!]^{u}$,
and similarly for the class *Husband*.

Again, $isPartOf_{1,2}$ are just arrow names without any formal semantics attributed to them, their semantics of real isPartOf-mappings is captured by declaring them identifiers for classes *Wife* and *Husband* together with some other their properties, for example, each of them covers the target class (note triangle heads of the arrows), we omit detailed discussion here. Well, "the husband of family A11" is a quite concrete person, say, John, and "the wife of family A11" is a another person, say, Mary, and thus we have the corresponding arrows into the class *Person*. The converse arrows are multi-valued (double arrow heads) because the same object John may well be "the husband of the family #A11" and (at another time moment) "the husband of the family #B55". Note also that attributes and methods of John as a *Person*-object and John as a H*usband*-object may be quite different, and this justifies the presence of three different classes on schema Fig.1b instead of one class *Person*.

On a whole, what is essential for us here is that all the semantic relationships can be expressed in the graphic language of (i) nodes denoting classes, (ii) arrows denoting mappings between them, (iii) diagram markers denoting predicate declarations (note, an arrow is nothing but a simple diagram and figure arrow heads/tails/bodies are nothing but markers hung on these diagrams). So, mathematically, the structural part of a business model is a collection of variable sets and mappings between them, together with some set of diagram predicate declarations. A natural syntactical presentation of such a model is a directed graph some diagrams in which are labeled by markers taken from a predefined signature. Such graphs are called *(generalized) sketches* (more accurately, S-sketches with S the name of the signature). However, this description is still incomplete and one more essential component must be added.

3.3. Finite presentations of infinite universes

Given a collection of sets and mappings between them, one can generate new sets and new mappings. For example, given sets *A* and *B*, we can take their union or intersection together with inclusion mappings $A \cap B \to A$, $A \to A \cup B$. Or, given a set *A*, one may form its powerset, that is, the set of all subset of *A*, **P***A*, together with their membership relation $M \subset A \times \mathbf{P}A$. The later is actually nothing but a set *M* equipped with two projection mappings $a: M \to A$ and $p: M \to \mathbf{P}A$ jointly having a special property ensuring that *M* and **P***A* behave themselves as needed. Or, for the universe presented on Fig.1b, one may compute a subclass *Marriage* of the class *Person* consisting of persons that are, or had been, married. Or, if there is an *age* attribute for the class *Person*, one may compute a subclass *YoungPerson* of *Person*-objects with *age* of less than 30. And so on, and so on.. These examples also show that operations on sets (and, in fact, mappings too) producing new sets (and mappings) are well studied in the database context under the name of queries.

Thus, normally, a business domain is an *infinite* universe of varsets and mappings, over which certain relations and operations are defined. In its turn, the task of business modeling is to build some *finite* and *complete* presentation of the universe. Here completeness means that any fragment of the universe can be recovered from the presentation by applying operations (queries) taken from some predefined set (query language). Note the algebraic nature of the construct: since operations are present, a business universe is an algebraic structure while the task of business modeling is nothing but a canonic algebraic task of finding a finite presentation for a given algebra. What was said above about the structural part of business models is valid also for their dynamic part – transactions changing the state of the system. Namely, a business model should specify some set of basic transactions from which any transaction of interest can be derived by consequent or/and parallel composition.

4. A bit of mathematics relevant for business modeling

4.1. Set universes, category theory and toposes

The construct of universe of sets together with operations and relations over them appeared as a subject of mathematical studies at the beginning of the century. More accurately, that time mathematicians were trying to figure out what is **the** universe of sets where the entire mathematics could be developed and properly formalized once and forever. As it often happens with searching *the-universes*, the problem was surrounded with various philosophical speculations but a fruitful idea gradually emerged: there is no “the set universe of mathematics” but, instead, many different set universes are possible. The idea became explicit in 1950s-60s and, finally, got a well manageable shape in category theory (CT) in 1970s. CT brought great flexibility not only in that explicated different set universes but also in that universes of different set-like objects, for example of variable sets, were built (see a popular presentation in [6]). Nowadays, for a category theorist, an arbitrary universe of arbitrary set-like objects is a quite ordinary mathematical structure like a group or linear vector space for a classical mathematician. Such a structure was called a *topos*, and now the topos theory is an active and one of the major divisions of CT.

A rich hierarchy of toposes of different kinds was built and studied in CT. An important peculiarity is that different toposes may have different underlying logics of reasoning about sets and their elements. In addition, the latter are expressed in algebraic terms as availability or not availability of certain operations over sets. For example, the powerset operation as it is described in section 3.3 is legitimate only for toposes where we do not care about operations’ effectiveness but will be illegal for effective toposes: indeed, the size of $\mathbf{P}A$ is exponentially growing with the growth of A. Thus, to summarize, different kinds of set universes are possible, each of them induces a certain logic and all the structure is treated in algebraic terms. The latter point has important operational consequences: algebraic models (of logic, universes etc) is the most immediate mathematical way to effective manipulations.

4.2. Toposes, sketches and business modeling

The previous section suggests that toposes are a natural mathematical framework for BM. However, topos theory in its current state of the art is not of much help for the practice of BM. The point is that CT-people prefer to work with universes as whole entities rather than with their finite presentations, and indeed, presentation-independent view is very useful for conceptual analysis and theoretical studies. However, for practical work one does need to deal with one or another particular presentation. (Similarly, modern geometry is impossible without the notion of vector and vector operations invariant with respect to coordinate systems but to perform a concrete geometrical calculation one needs to choose a particular coordinate system (the most suitable for the task!) and work with coordinate presentations of vectors and operations involved within this system).

Well, a presentation-oriented trend in CT is associated with the concept of *sketch,* which was directly intended for effective graphic presentation of mathematical structures (the book [BW99] presents a sketch-modulated account of CT). Unfortunately, classical CT-sketches cannot be utilized for BM in an immediate way (see a discussion in [5, section 1]) and need essential development towards their applicability in SE and BM. A suitable elaboration was performed in [5] where the concept of generalized sketch was proposed and developed (see also [2] for a short presentation).

Briefly, the machinery of generalized sketches is an immediate graph-based counterpart of ordinary algebra and logic as they are presented in textbooks on universal algebra and mathematical logic. It is a very flexible and powerful specification language invented just for specifying universes of set-like objects. A sketch immediately specifies a semantic picture rather than some intuitive picture for which semantics should be searched later (if ever, as it goes with ER, or UML or the like diagrams. In the topos-sketch viewpoint, all these languages can be treated as special visualizations built over basically the same common sketch format for specifying toposes).

5. Summary

Three main questions posted in introduction now may be answered as follows.

(1) Any given business domain *D* is (mathematically) a topos, that is, a particular case of a quite general mathematical structure. Probably, even the following more refined picture is valid. Each specific kind of business **B** (banking, insurance or telecom industry, etc.) determines its own kind of toposes, **Top(B)**, so that any business domain *D* in **B** is a topos of sort **Top(B).**

(2) Normally, toposes *D*'s are infinite and, given such a topos (domain) *D*, the task of business modeling is to find a finite yet complete presentation of *D*. Syntactically, this presentation is specified by a (generalized) sketch, that is, a directed graph with diagrams marked by labels taken from a predefined signature corresponding to **Top(B).** (In fact, setting **Top(B)** amounts to nothing but setting some signature of legitimate predicate and operations).

(3) So, thinking semantically, business specifications are sketches whatever visualization superstructures (ER, OMT, UML) were built over them. Then, a natural mathematical apparatus for managing and manipulating business specifications is the machinery of deriving and rewriting sketches. In essence, the latter is nothing but a counterpart of ordinary logical derivation and algebraic term rewriting for the graph-based situation.

References

[1] M. Barr and C. Wells, “Category theory for computer science”, CRM, Montreal, 1999

[2] Z. Diskin, B. Kadish and F. Piessens, “What vs., how of visual modeling: The arrow logic of graphic notations”. In: Behavioral Specifications in Businesses and Systems, Eds. H. Kilov *et al*, Kluwer Acad., 1999

[3] Z. Diskin and B. Kadish, “Variable set semantics for generalized sketches: Why ER is more object-oriented than OO”, Data and Knowledge Engineering (to appear).

[4] Z. Diskin, “Formal semantics for the UML”. In preparation.

[5] Z. Diskin, "Generalized sketches as an algebraic graph-based framework for semantic modeling and database design", University of Latvia, Research Report, M97-1, Riga, 1997
On ftp: //ftp.cs.shalmers.se/pub.users.diskin/REPORTS/UR-M97-1.ps

[6] R. Goldblatt, Topoi. The Categorical Analysis of Logic, North-Holland, 1984

[7] H. Kilov, Business Specifications: The Key to Successful Software Engineering, Prentice Hall, 1999