SCIENTIFIC PROCEEDINGS OF RIGA TECHNICAL UNIVERSITY
Series – Computer Science Applied Computer Systems – 4th Thematic Issue
2003

FORMAL INTEGRATION PERSPECTIVE IN THE SOFTWARE DEVELOPMENT

E. Asnina

Representation conflicts, integration, sketch, UML

1. Introduction

This paper is further observation of research results in sketch approach [1] and its applying in software engineering. The paper considers perspectives of sketch approach usage for formal integration of static data models.

The paper is divided into four sections. The next section explains why formal integration of data models is necessary and what kinds of problems can occur in an integrated data model. The section 3 gives a short sketch approach description, necessary for understanding of this paper content. The section 4 shows sketch approach applying for two conceptual UML diagrams integration. General step order of sketch integration and its estimation are given in the section 5. Possible further researches are described in the paper conclusions.

2. Necessity and problems of sketch integration

Now-a-days a number of software application spheres is growing fast. Information systems become more complicated, development time increases. In practice, it is not possible to create a whole model of a system at once.

Integration is necessary if designed information systems are large. In account of system size, semantic models need to be created from the viewpoint of system parts or of system users groups and then the parts need to be integrated in one model. This can lead up to differences of data structures and process of integration can become far from trivial task. Without that, a problem of developers groups communication exists, as it is necessary to handle model changes. A problem of development of large multidisciplinary systems where different discipline semantics meets exists, too [2]. Here, the independence of development and the differing cultures of the fields cause incompatibilities between models and programming interfaces. Large information systems require defining an optimal data structure, which could allow deriving necessary information already in computing process. However, the scheme integration is not a new problem, but with the advent of object oriented databases developers have met this problem again.

Existing problems in the scheme integration are the following [3]:

- Naming conflicts originate from using equal terms for different real world concepts (homonyms) or from representing the same real world concept using different terms (synonyms).
- Structural conflicts occur when different data model constructions represent the same information. This kind of conflicts can be found in (i) aggregation, i.e. when attributes or methods are missing; (ii) abstraction, i.e. when there are differences in specialization or decomposition; (iii) generalization, i.e. all other structural conflicts.

- Scaling conflicts occur if different scales are used for the same measure.
- Static-dynamic conflicts present at the object oriented approach occur, i.e. when the same information is designed as static property in one view and as dynamic property in another view. For example, *age* property can be stored as attribute in one view, but in another view, *age* can be computed with a method on basis of the birth date and the current year.

Usually for the integration of static structures one applies formalism of the first order logic, but it is too bulky for graph-based constructs. The usage of graph-based logic may be more comfortable. Therefore, the formal sketch approach can be successfully used for handling representation conflicts.

3. A short description of the sketch approach

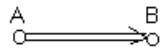
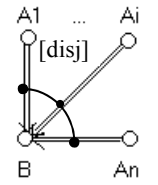
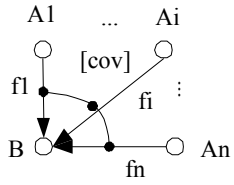
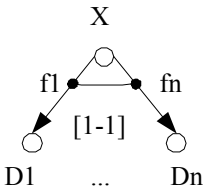
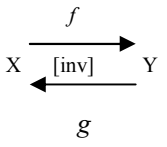
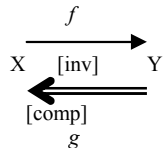
In the basis of the sketch approach are principles of category logic [4], [5], [6] and a special kind of thinking — the so-called arrow thinking [1], [7]. The basic idea underlying the approach consists in specifying any universe of discourse as a collection of objects and their morphisms. Objects have no internal structure: everything one wishes to say about them, one has to say in terms of arrows. It means object structure and behavior are encapsulated and accessible only through the arrow interface. The distinction from functional data modeling is that here we deal with categories, not with sets and sets elements, and constraints are hung on arrow diagrams, not on nodes.

In the arrow logic, specifications are oriented graphs, which consist of nodes and arrows, where some fragments are marked. These markers note predicates taken from some predefined signature. These graphical constructs are called sketches [4], [5], [6], [8], [9]. Sketches are noted as Π-sketches, where Π is the name of the signature of diagram predicates. For example, UML language [10] defines some sketch signature Π_{UML} so that every UML diagram D could be represented as a special visualization of the logic specification S_D of the corresponding Π_{UML}-sketch. In common, any diagram with precise semantics (to be described in mathematical terms) actually hides a sketch in a suitable signature of markers. Any given diagram property has a predefined shape, that is, a configuration of nodes and arrows for which the property makes sense. A diagram predicate is specified with its name and graph called the *logical arity shape* of the predicate (Table 1). The arity shape should be supplied with auxiliary graphic means like arcs or double-body arrows for visualizing predicate declarations on schemas. In order to declare a predicate P with some arity shape G_P for a system S of sets and functions, one must assign S-sets to nodes in G_P and S-functions to arrows in G_P in such a way those adjoinness conditions between nodes and arrows are respected. It gives wide possibilities for any signatures of modeling languages creating for diagrams formal converting into the sketch format and for the sketches handling in the completely formal way. Table 1 shows examples of predicates used in the paper. These are predicates of set inclusion (Is_A), disjointness, and covering, separating of a family of functions, inversion, and composition properties. The inclusion denotes UML language's association of generalization, the inversion denotes undirected association, and the composition is the special case of aggregation in UML.

In order to handle sketches in a formal way, the category logic offers the so-called diagram operations (data queries) over a sketch [6], [11]. Operations allow customizing sketches by extending them with derived items. A diagram operation F is specified by a sketch D_F,

denoting its interface in which a subsketch D_F^{in} of input data is designated, that is, an operation F is specified by inclusion $i_F : D_F^{in} \to D_F$ (D_F is called output sketch). The body of operation is a procedure [[F]], which calculates an extension of derived items of D_F from a given extension of D_F^{in}. Table 2 shows operations used in the paper.

Table 1. The list of diagram predicates

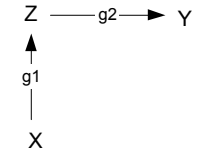
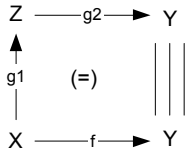
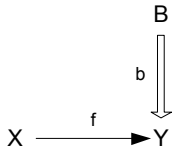
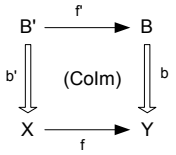
Predicate name	**Arity shape with visualization**	**Denotational semantics**
Set Inclusion - the source set is a subset of the target set and mapping is their inclusion.		$A \subset B$ *and* $f(a) = a$ *for all* $a \in A$
Disjointness - an element of the target set may be an element of the only one subset.		$\bigcup_{i=1}^{n} A_i \subset B$
Covering - each element of the target set is a value of at least one of the mappings.		$(\forall b \in B)(\exists i < n) b \in f_i(A_i)$
Separating family of functions - is somewhat dual to the covering predicate (in the category theory, this duality can be expressed in precise terms). It can be declared for a diagram, which consists of a source node and a family of arrows going out of it. We precisely express the internal tuple-structure of X-objects by declaring the corresponding property for some arrow diagram adjoint to the node X.		For any $x' \in X$, $x \neq x'$ implies $f_i(x) \neq f_i(x')$ for some i Note however that in such a case the tuple-function $f = \langle f_1 \ldots f_n \rangle$ into the Cartesian product of D_i $f = \langle f_i \ldots f_n \rangle : X \to D_1 \times \ldots \times D_n$, $fx = \langle f_1 x, \ldots, f_n x \rangle$ is injective (one-one) so that elements of X can be considered as unique names for tuples from a certain subset of $D_1 \times \ldots \times D_n$, i.e. the image of f.
Inversion		$(\forall y \in Y) f(g(y)) = y$
Composition is that the part can be the one of the only one whole.		$Y \subset X$ and $g(y)=y$ for all $y \in Y$, and $(\forall y \in Y)\ f(g(y))=y$

The operation of composition (the 1st row of Table 2) allows specifying a function, i.e. a result of composition of two arrows. The operation *CoImage* (the 2nd row of Table 2) allows deriving a subset of the set that satisfies some constraint. For example, the operation *CoIm(sex, M)* makes it possible to get derived item *MRead*, which denotes the subset of male sex reader (Figure 4).

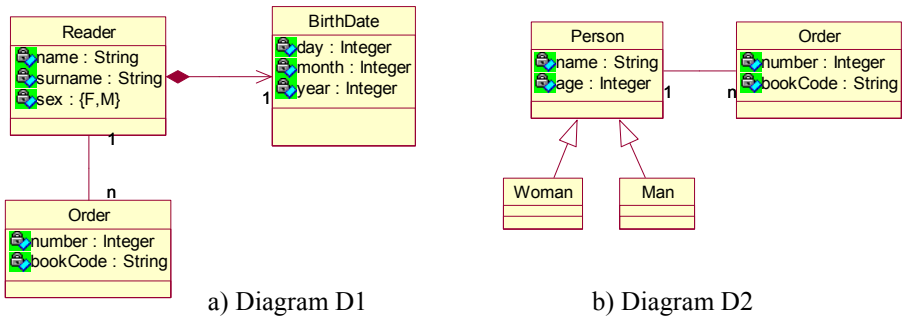
4. Sketches and correspondence equations usage for view integration

The sketch approach can be successfully applied for system views formal integration with possibility to solve structural conflicts between views, because even the simple replacement of labeling nodes by labeling arrow diagram allows avoiding certain kinds of structural conflicts between views.

Table 2. Diagram operations

Name (marker)	Arity shape		Denotational semantics	Linear notation
	Input sketch	Output sketch		
Composition (=)			$(\forall x \in X)\, f(x) = g_2(g_1(x))$	$f = g_1 \blacktriangleright\blacktriangleright g_2$
CoImage (CoIm)			$B' = \{x \in X : f\,x \in B\}$ f' = f restriction on B'	$B' = f^{-1}(Y)$ or else $B' = CoIm_f(B)$

Let consider an example of two UML class diagrams integration (Figure 1). Let suppose, that a program model, described in the UML language, is analyzed from viewpoints of library employees and of library information statistic calculations.



a) Diagram D1 b) Diagram D2

Figure 1. UML class diagrams

The first view is shown as a class diagram D1 (Figure 1(a)). Semantics is that some reader orders printed publications in the library. Readers are organized as a class *Reader* with attributes *name, surname,* and *sex*. The value of the *sex* attribute can be *F* (female) or *M* (male). But the predefined type of the attributes *name* and *surname* is `String`. Real world objects of the class *Reader* have a property "birth date". This property is shown as a class *BirthDate* with type `Integer` attributes *year, month* and *day* in the view D1. Between a class *Reader* and a class *BirthDate* exists an association of the composition. All the orders are united in a class *Order* with attributes *bookCode* (`String`), i.e. a library code of a printed publication, and *number* (`Integer`), i.e. a number of an order.

Class diagram D2 (Figure 1(b)) represents the second view of the problem. I remind you that this view represents the viewpoint of statistic calculations. Library visitors are organized into a class *Person* with attributes *name* (`String`) and *age* (`Integer`). To that, the class *Person* is specialized with classes *Man* and *Woman*.

A task is to integrate these two views, D1 and D2, in a formal way, overcoming structural conflicts between them.

In our example, conflict points are the following:

- Objects of *Person* and *Reader* classes are the same real world objects class (synonyms).
- The set of attributes of *BirthDate* class is the base information for *age* attribute of class *Person*.
- The subclasses *Woman* and *Man* of class *Person* represent the sex particularity of real world objects class, i.e. of library visitors. In the *Reader* class, this information is represented as the *sex* attribute.

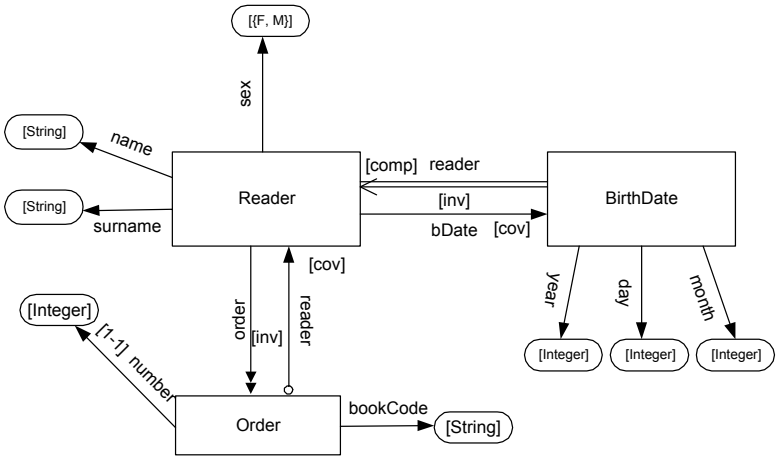


Figure 2. UML class diagram D1 converted into sketch S1

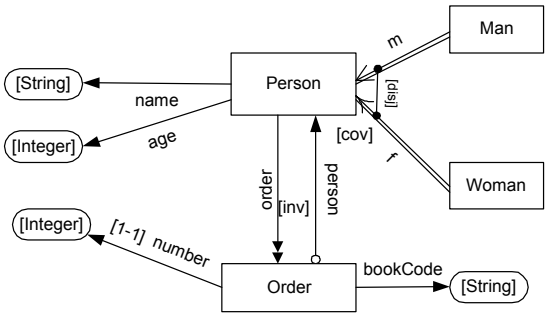


Figure 3. UML class diagram D2 converted into sketch S2

A more careful analysis of the situation shows that that part of the information considered as basic by the second view, can be derived from the first view (by making the corresponding queries). To specify this observation formally one must proceed as follows.

Firstly, let convert the UML class diagrams into sketches; i.e. into directed graphs, in which some arrow diagrams are marked in a special way (Figure 2 and Figure 3). Note that all attributes of the classes are represented as functions into nodes, whose intended semantics is predefined (and supported by the computer system). For example, the attribute *bookCode* of the class *Order* is represented by the function *bookCode* into a node labeled by *String*. Note also, that in the sketch approach labels *String, Integer* or *{F, M}* are only markers, which are hung on the corresponding nodes. I.e. constraints that impose on nodes intended semantic interpretation[1]. Otherwise, those are predefined data types in the sense of some programming language. To distinguish those type nodes from abstract [11] in the sketches, the latter are represented by rectangles, but the first by rectangles with rounded corners. Then it is necessary to specify correspondence between views in the sketch language (the second step).

[1] At the same time, *Reader, Order*, etc. are merely *names* labeling corresponding nodes without imposing any constraints.

Secondly, each original sketch must be extended (if it is possible) with nodes and arrows, which denote derived elements so that correspondence between views becomes explicit and could be formally described (Figure 4). At first, let extend the first view S1 with predefined computing function *(currentYear-*): Integer→ Integer*. After that, compose three arrows: *bDate, year,* and (*currentYear* - *)[2] in order to get the arrow *age'*. The corresponding diagram marker (=)' just expresses the fact, that the arrow *age'* is obtained by the operation of composition (Table 2). Then, the sketch can be enriched with two inclusion constructs (Table 1): *m: {M} → {M, F}* and *f: {F} → {M, F}*. These inclusions can be derived because the set *{M, F}* exists. After that, let apply two operations *CoIm* (Table 2). This type of operation takes, for the given function, the coimage of a given subset of the codomain in order to obtain the following derived items:

(MRead", mr", !")= CoIm(M, sex), (FRead"', fr"', !"') = CoIm(F, sex).

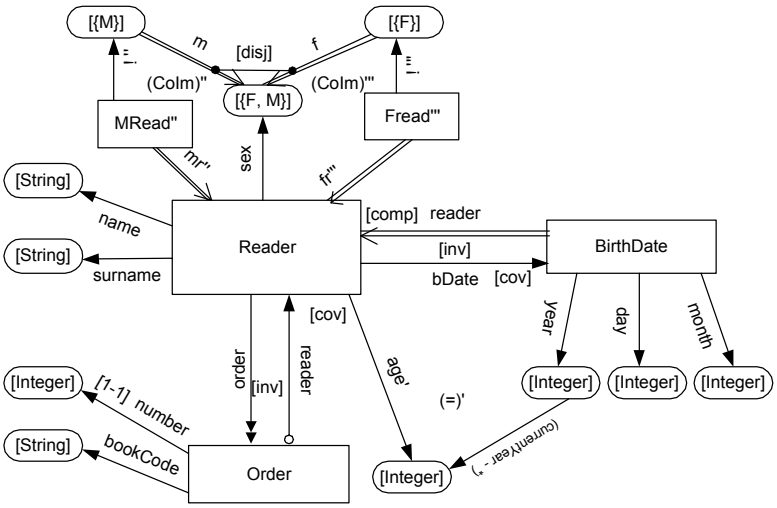


Figure 4. Sketch $\overline{S1}$ extended with derived items

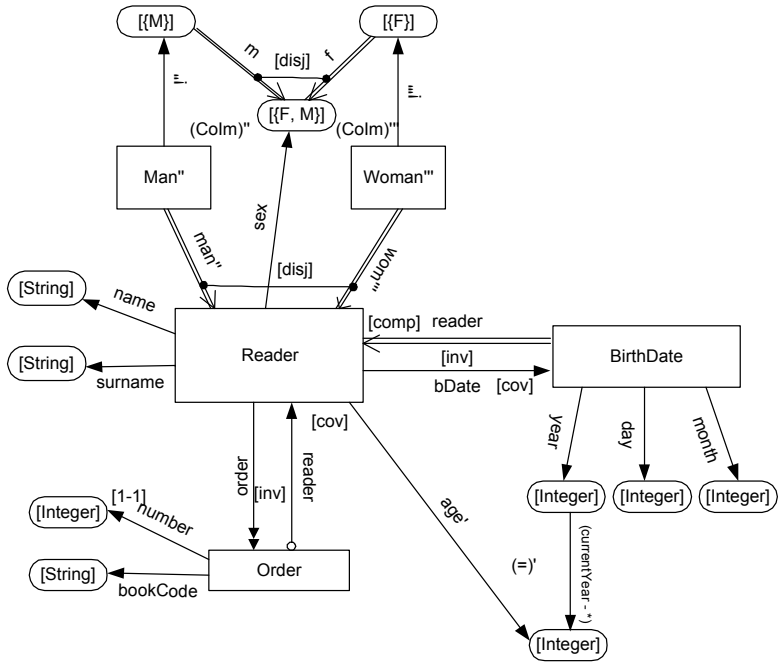


Figure 5. The full integrated sketch $\overline{SI} = \overline{S1} \oplus_{CI} \overline{S2}$

The *(CoIm)* marker hung on a square diagram just expresses the fact that the corresponding nodes and arrows of the diagram are obtained by the diagram operation *CoIm*. Consequently, the original sketch S1 is extended to the sketch $\overline{S1}$, in which some additional (but obligatorily derived) elements are specified (Figure 4). Sketch S2 hasn't been extended because there are no elements to be extended, so $\overline{S2}$ = S2.

Now, the integrating person (a designer or analyzer) can specify the correspondence between views in the form of equations. The equations fix identification of the corresponding nodes and arrows. Table 3 shows the set of equations E_{CI} (where *CI* is an abbreviation of

[2] Such superscripts as *, ', ", "', etc. allow to distinguish extended elements from base elements in the sketch.

"correspondence information"). This table denotes correspondence information only between functions (arrows), because the correspondence between functions means that corresponding domains and codomains of functions are equal too. The elements standing at the same column in the Table 3 give one equation, for example, $\overline{S1}.age' = \overline{S2}.age$, etc. So the correspondence between views can be specified in such formal way, and these correspondence equations have far reaching possibilities (e.g., in a case of disjointing an integrated sketch).

Table 3. The correspondence information between sketches $\overline{S1}$ and $\overline{S2}$

$\overline{S1}$	age'	name	mr"	fr"'	reader	order
$\overline{S2}$	age	name	m	f	person	order

The next step of the integration process is the creation of an *integrated sketch*. This step can be automated. The aim is to glue together those nodes and arrows that appear in the correspondence equations. Figure 5 presents the result of gluing and denotes it as $\overline{SI} = \overline{S1} \oplus_{CI} \overline{S2}$. The name *the full-integrated sketch* includes **all** the necessary basic information, and additionally some derived information. Mappings from the local sketches into the full-integrated sketch must be described (Table 4 and Table 5).

Table 4. The mapping a1 from sketch *S*1 into the full integrated sketch $\overline{SI}$

*S*1	sex	name	surname	order	reader	bDate
$\overline{SI}$	sex	name	surname	order	reader	bDate

Table 5. The mapping a2 from sketch *S2* into the full integrated sketch $\overline{SI}$

*S*2	name	age	order	person	m	f
$\overline{SI}$	name	age'	order	reader	man"	wom"'

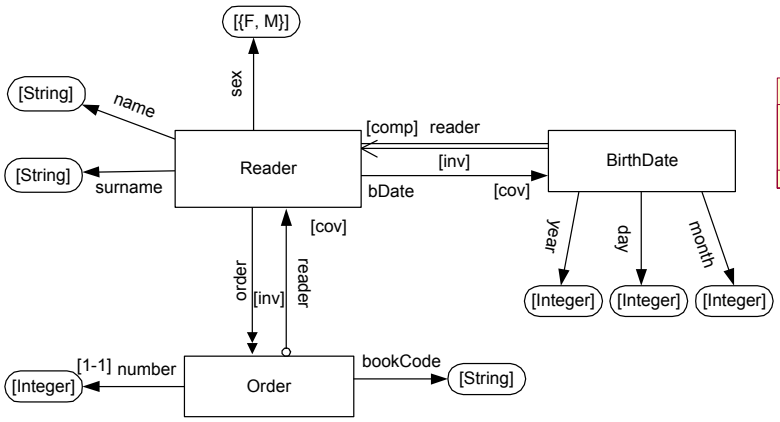


Figure 6. Generating sketch SI

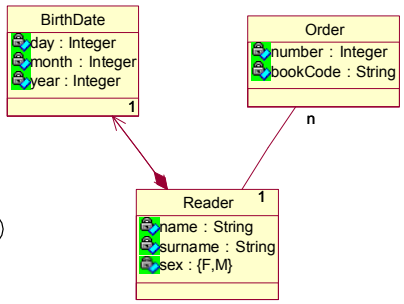


Figure 7. Integrated diagram

In order to finish the integration process an analyzer has to choose a subsketch *SI* of the full-integrated sketch so that all elements of $\overline{SI}$, which are not included in *SI*, could be derived from the latter by diagram operations from a prescribed list. Such sketches are called *generating sketches* (Figure 6). Table 6 and Table 7 represent mappings from sketch S1 and sketch S2 into the generating sketch *SI*.

Table 6. The mapping a1 from sketch S1 into generating sketch SI

*S*1	sex	name	surname	order	reader	bDate
SI	sex	name	surname	order	reader	bDate

Table 7. The mapping a2 from sketch S2 into generating sketch SI

*S*2	name	age	order	person	m	f
SI	name	bDate; year; (currentYear-*)	order	reader	CoIm(sex, M)	CoIm(sex, F)

To fix correspondence between views in the considered example it was sufficient to state correspondence equations over the initial sketches (i.e. between their basic and derived items), because the correspondence between views is the coincidence of extents of the corresponding items. Figure 7 shows the result of the generating sketch converting back into the UML class diagram.

5. Generalisation and estimation of the sketch integration approach

The example described in the section 4 presents a general approach for the integration of system views using possibilities of the sketch approach. Telling about large number of sketches to be integrated, correspondence information sketch S_{CI} [8] must be used instead of correspondence equations. In general, integration of n local schemes is disjoint merging of $n+1$ sketches ($S_1, \ldots, S_n, S_{n+1} = S_{CI}$) and refining the result using correspondences equations (i.e. gluing together certain items of the merge according to the E_{CI} equations). The result of the integration can be written as $\overline{S_I} = (\overline{S_1} \oplus \ldots \oplus \overline{S_n} \oplus \overline{S_{CI}}) / E_{CI}$, where $\overline{S_i}$ denotes results of extending local sketches with derived elements and $\overline{S_I}$ is the result of integration. Besides the integrated sketch itself — $\overline{S_I}$, the integration procedure determines mappings $a_i : S_i \to \overline{S_I}$ from local sketches into $\overline{S_I}$. Images of these mappings cover $\overline{S_I}$ so that each of $\overline{S_I}$ nodes and arrows belongs to $a_i(S_i)$ for at least one i.

So, the procedure of the diagrams (described in some modeling language) integration is the following: 1) Converting diagrams into sketches by some predefined list of predicates, 2) Extending sketches with derived items by means of some predefined list of diagram operations and determining correspondence equations, 3) Merging graphs of $n+1$ local schemes and refining the result according to the correspondence equations, that is applying a closure operation for the sketches, 4) The integrated graph converting into a sketch integrating diagram markers from the local sketches. However, here the marker integration problem can arise. The formal base of the diagram operations and predicates lists creation is described in details in [7], [11].

The paper [3] specifies the following criterions of integration methodologies: the used data model; the proposed strategies to solve representation conflicts; the approaches to handle redundancy removal; and the concepts to deal with scheme enrichments. The sketch approach for the integration process uses sketches. Any diagram, which has formal semantics, can be transformed into a sketch [7]. This approach reduces all kinds of representation conflicts between semantic views to two kinds of conflicts:

- Conflicts, in the base of which is conflict between basic and derived information. I.e. the information, considered as the basic in one view, could be considered as derived in other views.
- Constraint conflicts, i.e. conflicts between diagram markers.

The sketch approach proposes a gluing principle for redundant detail handling. It uses correspondence equations between sketches. This approach uses diagram operations that deal with new data constraints for scheme extending.

So, the sketch approach helps to integrate a structure of UML diagrams and diagrams described in some other modeling language in a convenient formal way.

6. Conclusion

The convenient graphical formal way of system views integration has been considered in this paper. This formal way uses the particularities of the sketch approach and corresponding information equations. The sketch approach proposes a common framework, which partially can be automated, for integration of different diagram kinds (and even for integration of different kind semantic diagrams). It reduces representation conflict kinds to two types: static-dynamic conflicts and constraint conflicts. In the paper, this approach has been applied for the integration of static structures of data models. But it can be applied for integration of dynamic structures (methods), too. Note, that this aspect of the sketch approach still requires additional researches, because the integration of methods has its own particularities.

The subjects of further research are sketch approach applying possibility for integration of data models dynamic structures and further integration methodology elaboration.

References

1. Asnina E., Osis J. *"Formalization Problems and Perspectives of The Program Development"* // In: Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, 3rd thematic issue, Riga, 2002, p. 145-156
2. Wheeler Th. J. *"Integration and Conceptual Modeling"* // In: Internet — http://people.cs.vt.edu/~edwards/RESOLVE2002/proceedings/Wheeler/
3. Eder J., Frank H. *"Schema Integration For Object Oriented Database Systems"* // In: Proceedings of the ETCE, Software Systems in Engineering, ASME, New Orleans, 1994.
4. Alksnis G., Osis J. *"Category Theory and Computer Science"* // In: Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, 2nd thematic issue, Riga, 2001, p. 59-67
5. Barr M, Wells C. *"Category Theory and Computer Science"* // Prentice Hall, London, 2nd ed., 1995
6. Alksnis G., Osis J. *"Formalization of Software Engineering by Means of the Theory of Categories"* // In: Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, 3rd thematic issue, Riga, 2002, p. 157-163
7. Diskin Z., Kadish B., Piessens F., Johnson M. *"Universal Arrow Foundations for Visual Modeling"* // In: Proc. Diagramms'2000, 1st Int. Conference on the theory and application of diagrams, Springler LNAI, No.1889, 2000 - pp. 345-360.
8. Diskin Z. *"Formalization of graphical schemas: General sketch-based logic vs. heuristic pictures"* // In: FIS/LDBD, web-page — http://citeseer.nj.nec.com/diskin95formalization.htm
9. Diskin Z., Kadish B. *"The Arrow Manifesto: Towards software engineering based on comprehensible yet rigorous graphical specifications"* // In: FIS/LDBD, web-page — http://citeseer.nj.nec.com/167037.html
10. Rumbaugh J., Jacobson I., Booch G. *"The Unified Modeling: Language Reference Manual"* // In: Addison-Wesley, 1999
11. Diskin Z. *"Generalized Sketches as an Algebraic Graph-Based Framework for Semantic Modeling and Database Design"* // In: FIS/LDBD, web-page — http://citeseer.nj.nec.com/diskin97generalized.htm

Erika Asnina, Riga Technical University, Meza 1/3, Riga, LV 1048, Latvia, B.sc.ing., as_erika@inbox.lv

Asņina Ē. Formālās apvienošanas perspektīva programmatūras izstrādē

Šajā rakstā ir aprakstītas problēmas, kuras var rasties datu modeļu apvienošanas gadījumā un viens no perspektīviem ceļiem datu modeļu apvienošanas jomā. Objektorientēto datu modeļu gadījumā apvienošanas procesu var sadalīt statisko struktūru apvienošanas un dinamisko struktūru apvienošanas daļās. Šajā rakstā tiks apskatīta pirmā daļa, jo dinamisko struktūru apvienošana prasa papildus pētījumus. Apvienošanas procesā parādās dažādi reprezentācijas konfliktu veidi: nosaukumu konflikti, strukturālie konflikti, mērvienību konflikti un strukturāli dinamiski konflikti. Šo problēmu pārvarēšanai ir piedāvāts skiču pieejas formālisms. Skiču pieeja pamatā atrodas kategoriju loģika. Rakstā apvienošanas soļu secība ir parādīta uz divu UML klašu diagrammu apvienošanas piemēra. Piedāvātā pieejā datu modeļi tiek pārveidoti skicēs, kuras pēc tam tiek paplašinātas ar atvasinātiem datiem un tiek apvienotas vienā integrētā skicē. Integrētā skice satur skaidri redzamu bāzes un atvasināto informāciju. Pilnīgi integrētā skice satur visus iespējamos bāzes datus un atvasinātus datus. Tas ļauj reducēt integrēto skici līdz optimālam stāvoklim, kad skicē ir atstāta tikai bāzes informācija. Beigās šo ģenerējošo skici var pārveidot atpakaļ datu modelī. Vēl viens skiču pieejas pielietošanas pluss ir tās spējas samazināt iespējamo konfliktu veidu skaitu.

Asnina E. Formal Integration Perspective in the Software Development

The paper describes problems occurring in the case of data models integration and one of perspective ways in field of data models integration. In the case of object-oriented data models, integration process may be separated in two parts, static structure integration and dynamic structure integration. In the paper, the first one is considered in account of that the dynamic structures integration requires additional investigations. Different kinds of representation conflicts appear during integration process. They are naming conflicts, structural conflicts, scaling conflicts, and structural dynamic conflicts. In order to solve these problems the formalism of sketch approach is proposed. In the base of sketch approach is category logic. The paper shows integration steps order on the example of the two UML class diagram integration. In the proposed approach data models are transformed in the sketches, which then are extended with derived elements and glued in one integrated sketch. The integrated sketch contains clearly visible base information and derived information. Full-integrated sketch contains all possible base data and derived data. It allows reducing integrated sketch up to optimal size, when only base information is left in the sketch. Finally, this generating sketch can be transformed back into the data model. One more plus of the sketch approach is its ability to decrease the number of possible conflict kinds.

Аснина Э. Разработка программного обеспечения: перспектива формальной интеграции

В статье описаны проблемы, возникающие в случае объединения моделей данных, и один из перспективных путей в этой области. В случае объектно ориентированных моделей данных процесс интеграции можно разделить на две части — объединение статических и объединение динамических структур. В статье описана первая часть, т.к. обьединение динамических структур требует дополнительного изучения. В процессе объединения возникают различные виды конфликтов представления информации: конфликты названий и измерения величин, структурные и структурно динамические конфликты. Для преодоления конфликтов подобного рода в статье предложен формализм эскизного подхода. В основе эскизного подхода лежит логика категорий. Последовательность объединения в статье показана на примере объединения двух диаграмм классов языка UML. В предложеном подходе модели данных преобразуются в эскизы, которые затем расширяются производными данными и объединяются в один интегрированный эскиз. Интегрированный эскиз содержит ясно различимую базовую и производную информацию. Полностью интегрированный эскиз содержит все возможные базовые данные и производные данные. Это позволяет сократить интегрированный эскиз до оптимального состояния, когда в эскизе остаётся только базовая информация. Окончательно этот генерирующий информацию эскиз можно преобразовать обратно в модель данных. Ещё один плюс применения данного подхода — это возможность сократить число видов возможных конфликтов.