# Precise Behavioral Specifications In OO Information Modeling


## Report by:

**Haim Kilov (Bellcore)**
**Bill Harvey (Robert Morris College)**

## Workshop Co-Chairs:

**Haim Kilov (Bellcore, haim@cc.bellcore.com)**
**Bill Harvey (Robert Morris College, harvey@rmcnet.robert-morris.edu)**
**Hafedh Mili (University of Quebec at Montreal, mili@aicha.info.uqam.ca)**


"De ene oorzaak van ellende is dat door verweving met (vermoede) economische belangen het veld van de slogans, reclamekreten, misleidende termen en valse beloften is vergeven: Expert Systems, Learning Machines, Teaching Machines, Automatic Programming, Higher-Order Languages, Visual programming, Program Animation, Software Engineering, het is allemaal humbug." (E.W.Dijkstra [9])[1]

## Workshop Purpose and Goals

The purpose of this workshop was to explore behavioral modeling in the context of object-oriented models, with an emphasis towards:

1) modeling the collective behavior of objects, with a particular interest in declarative constructs, and

2) modeling viewpoints, both along the lines of ODP's viewpoints (e.g., enterprise and information viewpoints versus computational viewpoint [1,5]), and in terms of different aspects of object behavior being of interest to different kinds of users within an organization (different domain experts, analysts, developers) or a la [2].

The presentations and discussions at the workshop provided both a snapshot of the state of the art and practice, including standardization activities, and an outline of open theoretical and practical questions that need to be addressed to advance the state of the art and practice of OO information modeling.

The object-oriented paradigm has solved many problems related to software packaging, but has created its own. With most OO modeling methodologies, especially data-driven ones, a functional glue has to be grafted back onto object models using additional constructs (e.g. various kinds of contracts). With traditional OO "A&D" methodologies, a dogmatic adherence to OO concepts as they are often used in programming precludes us from seeing different user viewpoints separately, and lacks the means to describe their interactions formally. The past few years have witnessed two emerging trends in OO modeling: a "refunctionalization" of data models, and a "subjectification" of objects. Blind faith in the object ideal is making room for healthy multi-paradigmatic cynicism (see e.g. [4] and activities related to collective behavior in generalized object models).

Our workshop builds on two previous— very popular!— OOPSLA workshops. All three workshops attracted wide international participation, including well-known OO experts. The first workshop dealt with the basics of information modeling concepts [6] and identified two major directions: unification of definitions and notations in behavioral modeling, by contrast to the relatively mature static modeling (no surprise there); and modeling inter-object behavior declaratively and early in the lifecycle. The second workshop [3] took up where the first left off, and the issue of viewpoints came up under different forms, some of which are addressed below. It also brought some bread and butter issues back on the table: how to acquire and express behavioral requirements (e.g., business rules) in a way that domain experts can understand and validate, and in a way that modelers can verify and map to an object information model. A number of participants expressed a strong desire to continue at OOPSLA'94, and this workshop had a more pragmatic flavor than the first two.

[1] The one cause of misery is that, because of (apparent) involvement of economic interests, the field abounds with catchphrases, advertising slogans, misleading terms and false promises: Expert Systems, Learning Machines, Teaching machines, Automatic Programming, Higher-Order Languages, Visual Programming, Program Animation, Software Engineering, all of it is humbug. (E.W. Dijkstra) [Translated by Ed de Moel.]

The following major conclusions of the second workshop provided an appropriate framework for continuity:

*Precise declarative (formal) definitions of concepts (like events, operations, roles, actions, triggers, etc.) and specifications of systems based on these concepts are essential for system specification understanding, validation, and unambiguous interpretation.*

*Collective behavior is essential for information system modeling. It should not be reduced to isolated object specifications.*

*An appropriate frame of reference is essential for understanding (e.g., business rules should be separated from computer-based implementation). Relations between frames of reference should be explicit.*

*Different modeling approaches should be reconciled using a small set of fundamental common concepts.*

The submissions to this OOPSLA'94 workshop show that precise and rigorous (if not formal...) behavioral specifications of collective behavior are being very seriously considered and successfully used in industry, both for requirement specification and for program development. This industrial experience is encouraging: as shown in the Proceedings of this Workshop [10], such diverse application areas as telecommunications, document management, financial applications, etc., as well as international standardization documents, successfully (re)use the same small set of fundamental concepts. These concepts provide a good framework for precise and abstract specification of the collective state (invariants) and collective behavior (pre- and post-conditions for operations) of several objects. "The greater concern with methodology is the consequence of the fact that computing science is one of the less knowledge-oriented branches of applied mathematics" [7]. The approaches discussed at this year's workshop will therefore help to substantially reduce the amount of reinvention[2] — and corresponding frustration!

The following quote set the stage for the workshop: "...many students don't want to be shown effective patterns of reasoning, they want to be told what to do. They have been trained to expect another mathematical cookbook, automatically read general guidelines as recipes that are supposed to suffice for the next exercise (something they - of course - rarely do). They expect a so-called complete methodology, with each next exercise carefully tailored to the potential of the preceding example and complain when they don't get what only the quack can provide. [We just addressed a bunch of industrial computing scientists, and the above phenomenon was alarmingly pronounced.]" (E.W.Dijkstra. Management and mathematics)

The organizers have structured the contributions into "Approaches", "Enterprise understanding", "Abstraction by specification", "Standards", "Documents", and "Applications". Each contribution was given 10 minutes (enforced!) to present the most important ideas. Questions and answers became a basis for discussion.

Although the workshop participants have quite different backgrounds, their presentations and approaches concentrated on a small number of the same important conceptual issues. What follows is an overview of these issues (and discussions), based on the perception of the workshop organizers. We don't even attempt to state that this is the only correct perception!

## Participants

APPROACHES: Doug Bryan (Stanford University); B. Cameron, C. Geldrez, A. Hopley, D. Howes, B. Mirek, and M Plucinska, (BNR Canada); John Daniels (Object Designers Ltd.); Bent Bruun Kristensen (Aalborg University); James J. Odell (James Odell Associates); David A. Redberg (AT&T Bell Laboratories); Charles Richter (Objective Engineering, Inc.)

ENTERPRISE UNDERSTANDING: Joseph Morabito (Merrill Lynch & Co., Inc.); Jim Ross and Tom Smith (CAP Gemini America,)

ABSTRACTION BY SPECIFICATION: Roger F. Osmond (Bytex); Dave Thomson (Object Technology International Inc.)

STANDARDS: Colin Ashford (Bell-Northern Research, Ltd); Erik Colban and Heine Christensen (Bellcore); Haim Kilov and Laura Redmann (Bellcore); Richard Sinnott and K. J. Turner (Department of Computing Science, University of Stirling)

DOCUMENTS: Lillian Cuthbert (Bellcore); Haim Kilov (Bellcore)

APPLICATIONS: Bill Harvey (Robert Morris College), Richard Price (Department of Veterans Affairs), and Cameron Schlehuber (Department of Veterans Affairs, VHA Database Administration); Augustin Mrazik and Jana Ceredejevova (ArtInAppleS spol. s. r. o); Stephen L. Nicoud (Boeing Computer Services); Kingsley Nwosu (AT&T Bell Laboratories) and Bhavani Thuraisingham (The MITRE Corporation)

[2] exemplified by a rephrased excerpt from an otherwise superb nameless paper: *the OO paradigm is excellent for modeling self-contained objects, but cannot be applied for modeling persistent relations... In this paper, we present a new model...* Fortunately, the situation is substantially better: quite a few papers in these Proceedings successfully apply the OO concepts for specifying persistent relationships; and an ISO standard [8] does just that!

# Activities

## Specifications

Specifications were the subject matter of the workshop. The participants discussed both the general properties of specifications and examples and lessons of their use in particular application areas. Some properties of a specification should be true for all viewpoints, e.g., internal consistency and correctness. Completeness (being able to answer all questions that could be reasonably asked[3]) is defined with respect to a viewpoint. In other words, only viewpoint-specific questions could be asked, and therefore the quantitativeness ("how precise...") is determined by the consumer's viewpoint.

It was noted (Redberg) that the work we're doing can be thought of as the analogy to software design patterns in the information perspective (see also [7, 12, 15]). Indeed, an information modeling construct is nothing more than an information modeling design pattern; it is reusable in application-specific models, it is an abstraction, it solves a real, general problem, and it precisely describes the thing (the objects participating in the relationship) and how the thing works (the behavior based on the relationship). [Unfortunately, not all software design patterns are defined in a rigorous manner!] Software patterns are used to architect software systems by abstracting code. Information modeling constructs are used to specify business information systems by abstracting (collections of) real-world entities. There exists an obvious need to relate the two viewpoints, and implementation libraries like the ones described in [12] can be used for this purpose.

## The need for abstraction

Many participants were quite unhappy with the current state of requirements. Several challenges were mentioned: requirements provided in terms of solutions; "thousands of use cases; and no common vocabulary" (Cameron et al.). As there is more to real world than just software (Redberg, Daniels, and others), the requirements need to be formulated in terms the customer can understand. Therefore constructs like message sending are not an appropriate way to write these requirements: they overspecify (Perhaps, they are as low-level as, in other contexts, **goto**'s, pointers, or hypertext links are [13].). The customer has to verify requirements because otherwise the system to be built may be quite different from what the customer wants. Therefore there is a need to clearly separate the concerns of the business enterprise from the concerns of the system development.[4] The fundamental purpose of a specification is to communicate intent. A good specification should be correct, complete, and be in the language of the consumer; and one sentence is preferable to 50 pages describing what the wrong thing is (Osmond). The specification should state what business processes and their effects really are, rather than describe 10,000 event traces (Richter).

Some customers prefer to present requirements in terms of external interfaces to currently existing systems. Although these presentations do not use system development terms, they reuse forms, charts, etc. from current legacy systems (Ross, Smith, and others). Obviously, this viewpoint on requirements does not properly represent the concerns of the business. As an example from the document management business (Kilov), paper and electronic document have quite different external interfaces, but solve the same business problems, and therefore the description of a business should not be based on a particular (legacy or otherwise) electronic document management system.

How do we keep users from presenting requirements in terms of solutions? The user often does design instead of analysis, and there is a need to distinguish between what the user says ("how") and what the user wants ("what"). There is also a need to distinguish a system for hire from a system for sale (Osmond). Analysis deals with things about which you have no choice (Daniels); design is everything else. These things have to be discovered, and walkthroughs (see [12]) are essential for this: they break people out of a box (Ross, Smith) by getting them into a meeting room where different viewpoints are merged. Sometimes different people get to know about each other as there is a lot of replication of like functions across different lines of responsibility (Ross, Smith, Osmond). Multiple projections from the same specification have to be used for multiple consumers (Osmond).

## Unrealistic expectations

"You can't push the button so that the system comes out the other side. And if you can then the user is left behind." (Ross, Smith). In other words, miracles don't happen: the domain expert, modeler, and developer have to think. Obviously, tools and object (collection) libraries help a lot, but in order to reuse a particular construct, its context has to be understood.

[3] "with a view to requirements modelling, the purpose of a model is to ask questions and demonstrate that answers can be given entirely in terms of the model." [16]

[4] The need to separate these concerns has been clearly stated, in particular, in the Open Distributed Processing Draft Standard: "Specifications and their refinements typically do not coexist in the same system description" [5].

## Precise declarative specifications

Many participants were very unhappy with lack of precision in lots of published work on OO methods. The need to specify behavioral semantics in a precise and explicit manner has been clearly recognized. Declarative specifications, including invariants for (collections of) objects, as well as pre- and postconditions for operations, have been presented and supported by many participants. These specifications have also been promoted by ISO standardization documents on Open Distributed Processing (ODP) [1,5] and General Relationship Model (GRM) [8,11]. All examples in the GRM use this approach (by presenting a specification in stylized English), and the users accept it "if you don't tell them that the specifications are formal" (Kilov, Redmann).

There exist several problems with this approach. Whereas it obviously makes a specification precise (and often quite explicit!), many customers and modelers may not be used to this kind of precision as there *seems to* exist a contention between formality and clarity (Thomson, Bryan, and others). Many customers (and modelers) will be more comfortable in using precise specifications translated from a formal notation like Z or Object Z into stylized English. Some people close up their minds when they hear about pre- and postconditions (Richter), but they usually accept stylized English specifications (Kilov, Odell). Almost all of us have encountered precise specifications in real life: think about legal documents, e.g.,, contracts like the one for buying a house (Kilov). Terse specifications are difficult for end users (Bryan), and so explanatory comments — as recommended by mature Z users, for example — may be very useful. Visual formalisms may augment or be otherwise used in the specification, provided that each element and relationship between elements of this formalism has been precisely and explicitly defined (Daniels, Redberg, Kilov). If these precise definitions do not exist — as too often happens — then a set of "cartoons" (diagrams) does not help much: we "need to annotate pictures with a little bit more formality" (Nicoud). And quite a few users understand that informal integrity constraints are difficult to enforce (Kilov, Redmann).

Another related problem deals with terminology. The same names may denote very different things ("altitude" has different meanings in different systems (Nicoud), and all of us know that more generic terms like "customer" are context-dependent). The approach taken by the Reference model for Open Distributed Processing provides a solution: a name is usable for denoting entities only within a particular, explicitly specified, context [5], so that the same entity may have different names in different contexts, and the same name may denote different entities in different contexts.

Still another problem deals with understandability. It is not sufficient for an understandable specification to be precise: even a Smalltalk specification may present major problems (Thomson)! Abstraction is needed as well, and precision is applicable at all levels of abstraction. A precise, but not abstract, specification is unmanageable and therefore not understandable. Eiffel has been successfully used for embedding extractable specifications (Osmond), in particular because it has language mechanisms for declarative contract specifications (pre- and postconditions, as well as class invariants). A human, not a machine, maps the specification onto an implementation. Understandability is as important — if not more — than verifiability (Richter). The need for a specification to facilitate communication among people (Thomson) has also been stressed, e.g., in [12, 15, 16].

A contract "happens" because a meeting of the minds is assured; the same is needed for creating specifications. Precision is needed for the minds to meet (not just in adversarial situation), and then the specification is created cooperatively, as a team effort (Kilov, Ross, Smith, Harvey, and others). As an example, in the Veterans Administration hospital information system, in order to document the intent, email messages leading to the integration agreement about the meaning of data are attached to this agreement (Harvey). A specification should provide a deterministic answer to user's questions, although at times the question may have no answer at that level. Different viewpoints may refer to different levels of abstraction. Different sets of questions are asked for different viewpoints. And finally, often the implementors have to "supplement" the specification.

Is a precise specification really that complicated? Consider Lewis Carroll's quote about compositions (Kilov, Redmann): "Alice had begun with 'Let's pretend we're kings and queens;' and her sister, who liked being very exact, had argued that they couldn't, because there were only two of them, and Alice had been reduced at last to say, 'Well, you can be one of them then, and I'll be all the rest.' " (*Through the Looking Glass*)

## Collections of objects

No object is an island (Kilov, Redmann, Kristensen, Daniels, Richter, and others). Most OOA methods do not deal with system-level functionality (Richter) or with properties of collections of objects. Early allocation of behavior to classes often promoted because of traditional OO programming constructs "is bad". There is a need to deal with relationships explicitly by providing a more abstract specification of aggregate behavior and

separating it from inner object behavior (Bryan, Richter, and others). Traditional specifications using attributes and isolated object operations ("object-centered method invocations" (Kristensen)) are not appropriate (too detailed) for understanding, although they may be quite precise (Kilov, Redmann, Redberg, Kristensen, Ashford, Bryan, and others).

Various approaches to dealing with collections of objects have been discussed. Collective state, for example, is specified in the ISO General Relationship Model using an invariant for a relationship. This approach is supplemented by specifying pre- and postconditions for operations applied to collections of objects (collective behavior). It has been described in more detail in [12] and has been successfully used to create understandable and reusable specifications (Kilov, Redmann, Redberg, Ross, Morabito, and others). In particular, the same generic relationships (composition, dependency, symmetric relationship, and so on) have been precisely defined and reused in very different application areas. For example (Redberg), an attribute-based model for telecommunications network and service operations (without easily understandable semantics) has been replaced with a much more understandable and reusable specification using generic relationships like dependency described in [12]. This model was provided both to users and developers. Thus, a library of sufficiently rich and expressive information modeling constructs can be specified in a precise and abstract manner, standardized (as in [8,11]) and successfully reused.

In another approach (Richter), a function is specified using initiator, outgoing signals, and pre- and postconditions (in pseudo-English): use English, but always talk in terms of object model constructs. Individual object behaviors may be specified later. Similarly (Daniels), behavior is described declaratively, by means of events and their order; an event has pre- and postconditions. Similarly (Odell), business rules are presented rigorously, in stylized English, using, e.g., operation constraint rules (pre- and postconditions), structure constraint rules (invariants), and so on. Similarly (Kristensen), transverse activities representing joint behavior have a directive and participants, may be classified (generalized and specialized), aggregated [e.g., pre-review, paper-evaluation, and post-review are aggregated into review-activity], etc. These activities represent idioms useful in analysis, design, and programming. Activities, functions, operations, etc., — and invariants! — jointly owned by several objects can be (if there is such a desire!)- implemented using messages attached to particular objects. This is not needed, however, at the specification level. Certain existing languages (like CLOS) permit implementation of collective behavior.

## Standards

Some ISO standards, such as ODP [1,5] and GRM [8,11], explicitly deal with specification of behavioral semantics. They define general concepts and constructs reusable in all application areas. They specify semantics in a precise, and often formal, manner. These specifications are written in stylized English, and some essential ODP specifications are formalized in [1] using such notations as LOTOS and Z.

These standards have been successfully used to specify the TINA-C architecture — reusable design patterns for broadband networks (Colban, Christensen). The logical framework architecture has been built upon ODP information, computational, and engineering viewpoints. However, the ODP information viewpoint had to be expanded because of the need to specify relationships, and therefore the GRM has been used in the information model. Invariants and pre- and postconditions have been successfully used in TINA-C specifications. Several ways of (not one-to-one!) mapping between information and computational models have been described.

The standards themselves have been discussed in some detail (Ashford, Kilov, Redmann, Sinnott), with a strong emphasis on the "why", i.e., on design considerations. The need for better understanding and for distinguishing between the ontology and representation (Ashford) has been stressed by all participants. Declarative specifications, such as invariants for defining a managed relationship (Ashford, Kilov, Redmann) and pre- and postconditions for the ESTABLISH operation for a managed relationship (Kilov, Redmann), provide a way to understand and therefore reuse on many levels, from concepts (such as invariants) to (fragments of) specifications. The participants stressed the need for formal descriptions of concepts and constructs, especially referring to the ODP approach (Sinnott) where Part 4 of the ODP Reference Model [1] provides the architectural semantics of ODP through the interpretation of basic modeling and specification concepts of [5] using various formal description techniques. It appears (Sinnott) that Z (or its OO extensions) is more appropriate for specifying the ODP enterprise and information viewpoints, whereas LOTOS — for the computational viewpoint.

## Documents

Domain experts, modelers (specifiers), developers, and others use documents in their work. These documents are quite complicated and need to be specified as well. In modeling a document collection, there exists a need to separate between the contents, logical layout, and

physical presentation (Kilov). To make documents understandable, the document contents model should correspond to the model of enterprises described by these documents. Existing technology (such as hypertext link management (Kilov) or SGML (Cuthbert)) solves some problems here, but is not adequate for specifying semantics of complex document collections.

## Part-whole

And finally, there was a heated discussion about whether a system behavior may be defined from its parts alone? It appeared that, in addition to parts, we need the description of the relationships between parts and the description of the whole as well. Example (Kilov): when a document is created out of components, perhaps components of other documents, not all document properties are defined by the components' properties or even by the relationships between them. The new whole is not just the sum of the parts (e.g., the title of a new document is not defined by its parts).

Given a "high-enough" level of components, could you build a system? The users require and describe capabilities, "parts". Can the specification of system behavior be defined out of specifications of component behaviors? (Things like relationship objects (and perhaps invariants) are also parts!) Perhaps... pushouts in category theory may provide an answer (some workshop participants have also been students at the category theory tutorial by Jose Fiadeiro and Tom Maibaum a day before and highly praised it).

## Conclusions

The workshop considered in detail how to meet the demand for a rigorous specification that corresponds appropriately to a form of presentation whose meaning can be validated by domain experts. Particularly productive was the discussion of the tension between the mode of customer requirements presentation and the need for the analyst to receive requirements that are not distorted by speculations on the part of the domain expert that transcend the expert's role and anticipate solutions. Dialog underscored the importance of correct treatment of the collective behavior of objects. We started the discussion on "part-whole", made some progress, and want to continue at OOPSLA'95.

## References


1. ISO/IEC JTC1/SC 21/WG 7, Information Technology - Basic Reference Model of Open Distributed Processing - Part 4: Architectural Semantics, International Standards Organization, August 1994. Working draft.

2. W. Harrison and H. Ossher, "Subject-Oriented Programming: A Critique of Pure Objects," in *Proceedings of OOPSLA'93*, pp. 411-428, ACM Press.

3. B. Harvey, H. Kilov, and H. Mili, "Specification of Behavioral Semantics in Object-Oriented Information Modeling: workshop report," *OOPS Messenger*, ACM Press, 1994. Addendum to the OOPSLA'93 Proceedings, pp. 85-89.

4. Geir Magne Hoydalsvik and Guttorm Sindre, "On the Purpose of Object-Oriented Analysis," in *Proceedings of OOPSLA'93*, pp. 240-255, ACM Press.

5. ISO/IEC JTC1/SC21/WG7, Basic Reference Model for Open Distributed Processing - Part 2: Descriptive Model (DIS 10746-2, February 1994).

6. H. Kilov and B. Harvey, "Object-Oriented Reasoning in Information Modeling: Workshop Report," in *OOPSLA'92 Addendum to the Proceedings*, pp. 75-79.

7. A.J.M. van Gasteren, *On the shape of mathematical arguments.* Lecture Notes in Computer Science, Vol. 445. Springer Verlag, 1990.

8. ISO/IEC JTC1/SC21, Information Technology - Open Systems Interconnection - Management Information Services - Structure of Management Information - Part 7: General Relationship Model. CD ISO/IEC 10165-7 N 8454. March 30, 1994.

9. E.W.Dijkstra, *Voorwoord.* EWD1156, 9 June 1993.

10. *Proceedings of the OOPSLA'94 Workshop on Precise Behavioral Specifications in Object-Oriented Information Modeling,* Robert Morris College, 1994.

11. ISO/JTC1/SC21 (Open Systems Interconnection, Data Management and Open Distributed Processing. OSI Management) P.Golick, H.Kilov, E.Lin, L.Redmann. U.S. National Body Comments on SC21 N 8454, ISO/IEC DIS 10165-7, General Relationship Model. Document number X3T5/94-198, July 11, 1994.

12. H.Kilov and J.Ross, *Information Modeling: an Object-oriented Approach.* Englewood Cliffs, NJ: Prentice-Hall, 1994.

13. H.Kilov. On understanding hypertext: are links essential? *ACM Software Engineering Notes*, Vol. 19, No. 1 (January 1994), p. 30.

14. B.Liskov and J.Guttag. *Abstraction and specification in program development.* McGraw-Hill, 1986.

15. C.Morgan. *Programming from specifications.* Second edition. Prentice-Hall, 1994.

16. Micheál Mac an Airchinnigh. Tutorial lecture notes on the Irish school of the VDM. In: *VDM '91. Formal Software Development Methods.* Lecture Notes in Computer Science, Vol. 552, Springer Verlag, 1991, pp. 141-237.