Object-oriented Behavioral Semantics

With an Emphasis on Semantics of Large OO Business Specifications

Haim Kilov
Merrill Lynch
Operations Services and Technology
World Financial Center
New York, NY 10080-6105, USA
Haim_Kilov@ml.com

Bernhard Rumpe
Institut für Informatik,
Technische Universität München,
80333 Munich, Germany
rumpe@forsoft.de

Ian Simmonds
IBM T J Watson Research Center
30 Saw Mill River
Hawthorne, NY
10532, USA
isimmond@watson.ibm.com

The workshop took place on Monday, October 6^{th}, 1997. With 23 accepted papers of good and high quality, written by 43 authors, quite a few new and consolidated ideas could be presented and discussed. Names and affiliations of the participants physically present at the workshop are included at the end.

Business specifications are used to understand and describe businesses independently of any computing systems used for their possible automation. This understanding has to be expressed in a simple, clear, precise, and explicit way, in order to provide the essential common ground for business domain experts and software developers. In order for specifications to be understandable, they have to suppress irrelevant details (i.e., be abstract) and should not be presented in terms of possible or existing solutions. It follows that, for example, business specifications do not have to provide an owner for system state or behavior (as in message passing): such owners are required by legacy OO approaches to system development which have nothing to do with business specifications.

Usually, operational specifications are less than perfect, as shown by the following operational specification provided by the Professor and exposed by the analysts – Sylvie and Bruno:

> "The difference between 'convenient' and 'inconvenient' is best explained by an example," said the Other Professor, who had overheard the question. "If you'll just think over any Poem that contains the two words--such as--"
>
> The Professor put his hands over his ears, with a look of dismay. "If you once let him begin a Poem," he said to Sylvie, "he'll never leave off again! He never does!"
>
> "Did he ever begin a Poem and not leave off again?" Sylvie enquired.
>
> "Three times," said the Professor.
>
> Bruno raised himself on tiptoe, till his lips were on a level with Sylvie's ear. "What became of them three Poems?" he whispered. "Is he saying them all, now?"
>
> "Hush!" said Sylvie. "The Other Professor is speaking!"
>
> Lewis Carroll, "Sylvie and Bruno"

Precise specification of semantics — as opposed to just signatures — is essential not only for business specifications, but also for business designs and system specifications. In particular, it is needed for appropriate handling of viewpoints which exist both horizontally — within the same frame of reference, such as within a business specification — and vertically — within different frames of reference. A (new or existing) complex system may be considered, on the one hand, as a composition of separate viewpoints, and on the other hand, as an integrated whole, probably at a different abstraction level.

The aim of our workshop — which continued the tradition of the five successful OOPSLA workshops and an equally successful ECOOP 97 workshop [5] on behavioral semantics — was to bring together theoreticians and practitioners to report on their experience with making semantics precise and explicit in various OO specifications. The Proceedings [6] of this workshop are available through the organizers. We think our workshop was a success.

The first four OOPSLA workshops on behavioral semantics led to a book [3]. Some participants of our workshops contribute to relevant national and international (ISO) standards (e.g., in Open Distributed Processing) and to various OMG documents [1,2,4], so that the workshop discussions and results will have a substantial influence on industry standards.

We present very short summaries of the talks, the raised issues and the discussions. Some material was shortened or left out as it became part of the final conclusions presented at the end of this paper.

Haim Kilov introduced the topic with two quotes: from Lewis Carroll (*Sylvie and Bruno*) on anthropomorphism, and from Alan Kay (*SIGPLAN Notices*, Sept.1997), a keynote speaker of OOPSLA'97, on the value of understanding the domain and finessing problems as opposed to solving problems.

Angelo Thalassinidis in a joint paper with **Ira Sack**, *An Information Modeling Approach to an Epistemic Typology*, gave some insights about the difficulty of formulating business strategy without being aware of the lessons of epistemology: the study of knowledge itself. When it is not used in business, "we keep misunderstanding each other and making the wrong assumptions". Modeling (meta-level) knowledge (e.g., "everybody knows that everybody knows") is important for business specifications. Complex knowledge constructs can be composed using simple information modeling constructs (Reference and Subtyping). This is easily understandable and may be applied in, e.g., "knowledge environments", by "knowledge workers" dealing with distributed knowledge and with issues like "How can I prove that my organization has the knowledge that it needs?". *Discussion:* everyone agreed epistemology is useful; we need to be explicit and acknowledge it. A lot of papers in, for example, the *Harvard Business Review* are about these issues (perhaps not explicitly enough). Some people may know but be unable to articulate!

Stuart Kent in a joint paper with **Kevin Lano**, **Juan Bicarregui**, **Ali Hamie** and **John Howse**, *Component Composition in Business and System Modeling* presented a precise approach for constructing compositions. He emphasized that it's important how to manipulate *specifications* – rather than executables. In particular, in view integration there may exist overlapping components. If they are executables it's impossible to merge them. Specifications help. (You have a lot of invariants to tie it together.) He showed how to use black box as well as glass box compositions combined with the techniques of UML and Catalysis and a formalization based on Larch. There is a need to check consistency between different models, as well as specific models against generic models. Also, there is a need to look for a component that satisfies "what I need". Cheap solution – repository of *specific* models. Three approaches to semantics to support it: Object calculus, different logics, category theory (Imperial College); Larch (Brighton); Pictorial (Kent himself) [semantics in diagrams] – restricted subset of first order logic. Kent also strongly advocated explicit refinement techniques and tools. Components are very tiny as the authors work in academia. *Discussion*: You need the specification of "the system as a whole" before treating components! Are tools useful or good? Perhaps not, as they may encourage developers to skip thinking, and immediately start to code, with trial and error. However, for large specifications tools are very helpful.

Karl Hoech and **Gary Daugherty** raised issues about improving the quality of software designs in their paper *Assessing the Quality of Formally Specified Types and Classes*. The goal was to provide hard and objective criteria instead of soft and fuzzy ones (pre- and postconditions were used extensively). Coupling and cohesion between classes have been examined using an extended and adapted version of the Embley-Woodfield criteria, where classes define ADTs. It was noted that multiple abstractions for the same class are prevented by (classical) OO. The challenge "bring a design that violates our criteria" still stands (could not find such a design yet). Q: have you applied it to the Gang of Four design patterns? A: to some, our guidelines stand pretty well. Q: The process you use may perhaps be used to identify frameworks. The concern was expressed that this approach could only work for toy examples.

Angelo Thalassinidis gave some interesting insights in his joint paper with **Ira Sack** *An Ontologic Specification of 'Strategic Signals*. Ontology is about the real meaning of certain things. He applied this theory to strategic signals – announcements that something very important is going to happen. Strategic signals are those that our opponent is not expecting us to take: threats (harm us and the opponent) and promises (harm only us). Threats are substantially stronger than promises. His goal is to describe signals in the business context, again using information modeling. What makes a signal credible? Both presentations by Thalassinidis were very interesting "hands in your pockets presentations", without slides. The *discussion* was about ontology being essential, and about the need for a "*Reader's Digest* epistemology" (i.e., for general use by non-specialists).

Allan Ash and **Haim Kilov** in the presentation of their paper *How to ask questions: Handling complexity in a business specification* gave some insights into the difficult task of getting the right information from the customer, and of presenting the result in small deliverables with high information content, visibility and clarity to ensure that specifications will be read. This was done for a moderately complex piece (accounting) of a system for Merrill Lynch. "Everyone glazed over" at accounting because the domain was perceived to be dull and overly complex (requirements of 107 users). Using precisely specified generic concepts and constructs brings a lot of time-consuming decisions up front: "do things right the first time". Resolving ambiguities is not terribly easy, but well worth it. Importance of articulation was stressed time and again. They avoided the word "object". They concluded that precision should be sought before correctness, structure before content, and precise specifications do not have to be detailed. *Discussion*: Q: Are state diagrams useful and when? A: Business domains go first ("invariants first, operations later"); but invariants may be motivated by operations ("generalize from bottom"). State diagrams are too detailed, you can become lost. Users prefer to think and explain systematically rather than linearly or locally; state diagrams if used have to be about a collection of things, not one thing. Emergent properties appear in invariants – how can they be presented in state diagrams?

Ilia Bider in his joint work with **M. Khomyakow** *One Practical Object-Oriented Model of Business Processes,* presented an object-oriented meta-model which allows the definition of *organizational* objects that capture and maintain sequences of stages of a business process. A sequence of stages can be used for undoing changes as well as reporting the history of a business process. The framework also allows the storage of future steps that are scheduled, but not yet done. The extended example of a deal, with deal state (showing what should be done independently of what happened before), and deal plan, was used. "Standard behavior" for a deal does not always work due to different kinds of events (e.g., check arrives "too early"). The definition is never complete, and the user can make manual changes. *Discussion*: This is a very nice illustration that the only essential business rules are those in contracts, laws and regulations; all others can be overridden by someone in the business. Business processes are about work simplification and efficiency for people who are still (or may always be) learning the business, and not necessarily about how it must be done. That is, work flows are part of (recommended, supportive) business design rather than (governing, essential) business specification.

Roger Burkhart described in his paper *Schedules of Activity in the Swarm Simulation System* a simulation model for concurrency. In modeling requirements for complex systems, the computer becomes the laboratory testbench, and very precise specifications are needed to meet the usual criteria of scientific experimentation. Decomposition in "complex dynamics" is not easy. Swarm (a collection of objects, together with a schedule of actions over these objects, with local clock) enables a population of agents to interact and produce the resulting behavior. Behavior appears from bottom-up (abstraction?). Concurrency is a challenge. The real emphasis is on events. Non-determinism comes through unconstrained orders of events. Schedulers can be rather freely defined to process events. Hierarchical compositions of swarms, reflective swarms, and observer swarms are possible. The developed event- and activity-orientation supplements object-orientation. The author's compositional techniques go beyond the traditional to express "the total composition of what's being done". Swarm is publicly available, uses dynamic and multiple classification (Objective C is used). Challenges of expressibility: distributed state under concurrent update and access; partial specification; modeling and computation within a system. Biological and social systems are not fixed and don't have fixed behavior.

Marc Shafer presented *Experiences Related to Integrating Information Modeling with the Business Requirement Definition Process* ... He used information modeling to "pin down business requirements". He notes two major problems: business requirements are too often stated in terms of implementation; and there are ambiguities in business requirements. "All your terms are to be 5 or 6 words long because you have to pin down the context" (e.g., the meaning of such terms as *claim* or *loss* is highly context-depend-ent). Several real-life examples show business requirements expressed in terms of solutions and, after asking relevant questions, transformed into require-ments in terms of the business domain. Pre- and postconditions and invariants help the business people to articulate what they are doing. Precise specifications of, e.g., *loss event exposure*, or *financial exposures*, can be formulated only by means of precisely specified relationships using information modeling. This is a formalized way to determine and document business rules; however, there still exists "a lot of kick back from traditional data modelers". The same approach can successfully be used to specify service environments where Reference and Subtyping relationships help to understand the situation.

Laurence Phillips presented joint work with **Stephen Bespalko** and **Alexander Sindt** *State-space Covering Strategies for Guaranteed Proper Termination*. Mission surety is exhibited if and only if the system accommodates its entire input space [including garbage input]. The strategies lay in finding partitions of the complete state space, so that for each partition, it is easy to ensure termination. Instead of considering each state, consider state properties (that is, characterize states by invariants). "When have I done enough to give myself a nice system?" The metaphor used is catching an elephant by enumerating all places where it is not, and looking at all other places. Conclusion: formal attention to the *full* state space before you write code leads to more robust systems.

Mark Saaltink in joint work with **Bran Selic** *A Framework for Behavioral Specifications* used (extended) state machine descriptions to specify some part of a system's behavior. Precise specifications of (a part of) the structure and behavior of object applications are a small but vital part (5 out of about 100 pages) of the UML semantics document. Their approach aims to be a proposal for tool semantics. The framework is based on continuously and discretely changing variables, thus allowing the provision of a precise semantic foundation for continuous activities in states, structured states, composite actions, and multiple views (e.g., of StateCharts). The resulting general model uses states as a fundamental concept and derives the event notion from that. The authors strongly advocate the use of state machines not only as implementation description but as one of many specification techniques (e.g., as a definition of the meaning of various kinds of interaction diagrams). (They did a non-trivial state machine model encompassing discrete and continuous change in 2 weeks.)

Bernhard Rumpe presented some ongoing work of his group with **Ruth Breu**, **Radu Grosu**, **Christoph Hofmann**, **Franz Huber**, **Ingolf Krüger**, **Monika Schmidt** and **Wolfgang Schwerin**, called *Exemplary and Complete Object Interaction Descriptions*, where a considerable subset of Message Sequence Charts (MSC) are given a precise semantics. Methodological guidelines were presented that propose four steps for using MSCs: starting from a comprehensible set of exemplary MSCs through generalization, use of repetition, alternatives and hierarchical structuring, the set of MSCs is completed. These complete descriptions of system runs are then broken down to state machines that describe the behavior of individual objects. A formal, integrated semantics of the notation is necessary to allow the definition of proper context, conditions, transformations etc. to ensure the correctness of such methodological steps. But this semantics – essential for tool vendors – need not be formally presented to the users of CASE tools.

Gary Daugherty in his paper *State Diagrams as Views of formal OO Models* integrated state machines with formal specifications. State machines are mapped to formally defined types, and subtyping is also included. States are defined through state predicates. The goal is to have a bidirectional mapping between state machines and formal specifications (and thus have a choice of analysis tools). Declarative style is easy to support. All transitions are treated as predicate transformers. During *discussion* it emerged that there is a difference between retrieving a state machine from a lower level description (code), and designing it from a higher level because some information is not present in the code (e.g., from the code you may only derive the minimal invariant, not the intended invariant of the developer).

Neelam Soundarajam in his paper *Interaction Refinement in the Design of OO Systems* introduced a formal way of defining interaction refinement in OO systems. This allows refinement of one interaction by a sequence of interactions (an invariant is defined to be satisfied by all such interaction sequences). Different refinements of the original specification are possible, and some may introduce additional objects. The refinements should be recorded, so the rationale behind the design is not lost. A refinement is proper if a particular relation holds between the source and target of a refinement. Discussion: Can this relation be called a "refinement invariant" by analogy with the "implementation invariant" from C. A. R. Hoare.

Robert France in joint work with **Andy Evans** and **Kevin Lano**, *The UML as a Formal Modeling Notation*, overviewed different approaches to give UML a formal semantics. This is essential since you may believe you understand "it", but your understanding may not be the same as the one of the authors'. "We spent an afternoon discussing the difference between a relationship and aggregation; 'here's an example that supports my view'; 'and here's an example that supports my view' ". Another approach is needed, or else a notation cannot be reasonably used. France identified three approaches: to supplement UML by formal notations, quite like Syntropy; to extend formal notations with object-oriented concepts, like Object-Z or SDL; and to develop a formal model from an informal model. The third approach is the most promising. Formalization helps to uncover problems with models and modeling notation. In the authors' framework, core concepts are at first informally defined, then formalized, and semantic composition and refinement rules added. Some issues: what are the core concepts to be formalized; how to compose the interpretation of a model; how to gauge the appropriateness of an interpretation; how to use different formalisms; and how best to present formalized semantics to non-formalists.

Michael Werner presented his approach to identify traversal paths (access paths) within partial class diagrams, called *Visitors, Access Paths and Semantics*. He defined the notion of a visitor pattern, and showed how to add visit operations. This both makes visibility explicit and constrains it. Actors get access to objects they want only through access paths (e.g., such visitors as customers can reach only a limited amount of information; auditors – a much larger amount). Traversal is like a view. (Motivation: view mechanism in relational databases.) An application is a composition of traversals and visitors. Shadow objects are available for the needs of the traversal initiator (to limit visibility). When the schema changes, one can update traversals, but use the same visitors. Pre- and postconditions are placed on visitations of objects along the path. Q (Miller): does it look like "negative inheritance"? A: yes.

Siobhan Clarke presented some very early work on composition in a large scale system development in her paper with **John Murphy**, *Developing a Tool to support Composition of the Components in a Large-Scale Development*. Through the use of aspect-oriented programming, components given by functional design, functional code, and black box design are plugged together. Aspect-specific code relies on the functional design or code. For functional design, Clarke wants to use UML, while for describing aspects she wants an appropriate aspect language. "Don't work with confusing application code."

Joaquin Miller presented the results of an email discussion (with **Haim Kilov**, **Peter Linington**, **Kerry Raymond**, and **Bryan Wood**), *Types, invariants and epochs: specifying changes in RM-ODP and ODP information language*, about the interpretation of RM-ODP concepts of types, invariants, static and dynamic schemas, and epochs. In particular, a dynamic schema defines the set of transitions from one __ schema to another __ schema; what is __? This may lead to changing somewhat the RM-ODP definition of a static schema. The paper also described how to use epochs to specify invariant changes ("always" for an invariant schema may be interpreted as the epoch for which this schema applies, so that epoch changes occur less frequently than changes specified in the dynamic schemas), and how to compose epochs yielding a new epoch at a different abstraction level. Not all state changes should be interpreted as type changes, but only those that are "of interest" in a particular context. The authors concluded that RM-ODP does not always answer questions, but provides a great way to ask them.

Joaquin Miller then presented another paper: *Help! How to specify policies?* Purpose, scope and policies are the subject matter of the enterprise viewpoint of RM-ODP. Purpose and scope have been defined. However, policies require deontic logic. They are defined using contracts which specify obligations, permissions and prohibitions — the subject matter of deontic logic. However, there are quite a few paradoxes in deontic logic, some of which were demonstrated in the paper. Miller raised an interesting question (to be answered by logicians): Is there a system of deontic logic that meets our needs? He referred to the Zeroth International Conference on Practical Systems of Logic for Policy Specification (http://enterprise.shl.com:80/policy/). Kent proposed to include temporal aspects into deontic logic (see, e.g., papers by Imperial College authors).

Many issues were discussed, resulting in the following unanimously accepted *conclusions*. They are very compact and perhaps may lead to somewhat different context-dependent interpretations.

- Start from top
- Discover from bottom
- Precision before correctness
- Do not confuse tool use with thinking
- Properties of a complete state space lead to an invariant
- Articulation is essential:
    - "All your terms have to be 5 or 6 words long because you have to pin down the context."
    - Use ontologies including relationships other than subtyping, to ask explicit questions about context
    - Be formal, but don't insist on exposing it
- Separate business from system specifications.
    - In code, separate business from plumbing.
    - Business rules, even detailed, should not be provided by developers
- Have a bidirectional mapping between graphical and formal specifications
- Abstraction (including selection of "appropriate refinement") has to be done by humans
    - Refinement invariants should include "relevant concerns" explicitly
- Open systems change their specifications

The diagram given below shows how to use formal semantics. While the formal semantics of a notation (representation which may be graphical or not) is needed to understand the notation, the formalization need not be exposed to the user. Formal semantics acts as a justification for the given transformation techniques that work on the (graphical) notations. The *commutativity* of the diagram ensures, in particular, the correctness of CASE tools (provided they are more than just mere editors).

The participants' affiliations in alphabetical order:

Allan Ash, Software Arts Inc., 70277.3315@compuserve.com.

Ilia Bider, Ibis Soft, ilia@ibissoft.se.

Roger Burkhart, Deere & Company, rmb@sanatfe.edu.

Siobhan Clarke, Dublin City University, sclarke@compopp.dcu.ie.

Gary Daugherty, Rockwell/Collins, gwdaughe@collins.rockwell.com.

Robert France, Florida Atlantic University, robert@cse.fau.edu.

Karl Hoech, Rockwell/Collins, kfhoech@collins.rockwell.com.

Stuart Kent, University of Brighton, Stuart.Kent@brighton.ac.uk.

Haim Kilov, Merrill Lynch, haim_kilov@ml.com.

Joaquin Miller, MCI Systemhouse, miller@shl.com.

James Odell, James Odell Associates, jodell@compuserve.com.

Laurence Phillips, Sandia Nat'l Laboratories, lrphill@sandia.gov.

Bernhard Rumpe, Munich University of Technology, rumpe@forsoft.de.

Mark Saaltink, ORA Canada, mark@ora.on.ca.

Bran Selic, ObjecTime, bran@objectime.com.

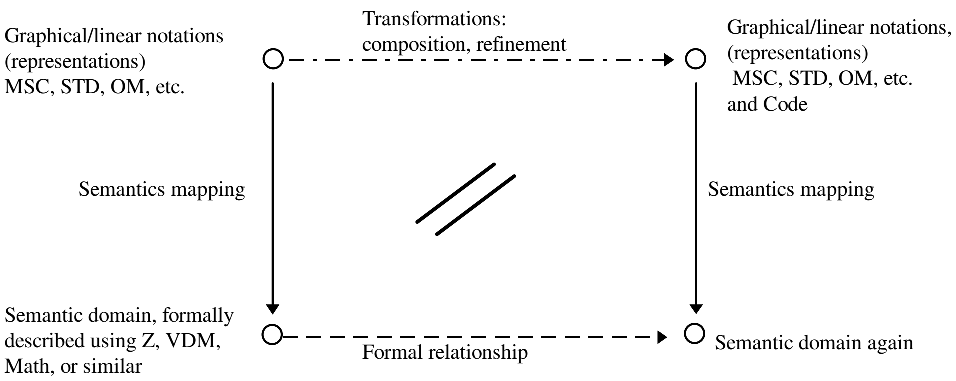
Mark Shafer, USAA, Mark.Shafer@usaa.com.

Ian Simmonds, IBM Research, isimmond@watson.ibm.com.

Neelam Soundarajam, Ohio State University, neelam@cis.ohio-state.edu.

Angelo Thalassinidis, MCI Systemhouse, athalassin@shl.com.

Michael Werner, Wentworth Institute, wernerm@wit.edu.



References

1. ISO/IEC. Open Distributed Processing - Reference Model: Part 2: Foundations (IS 10746-2 / ITU-T Recommendation X.902, February 1995).

2. ISO/IEC. Information Technology - Open Systems Interconnection - Management Information Systems - Structure of Management Information - Part 7: General Relationship Model, ISO/IEC 10165-7, 1995.

3. *Object-oriented behavioral specifications*, edited by Haim Kilov and Bill Harvey, Kluwer Academic Publishers, 1996, ISBN 0-7923-9778-9.

4. OMG Semantics Working Group Green Paper. OMG Document number ormsc/97-06-10r (Haim Kilov and Kevin P. Tyson).

5. 11th European Conference on Object-Oriented Programming, *Workshop on Precise Semantics for Object-Oriented Modeling Techniques*, Jyväskylä, Finland, 9-13 June 1997; Editors Haim Kilov and Bernhard Rumpe; Proceedings of Munich University of Technology, TUM-I9725, May 1997

6. Object-Oriented Programming Languages and Applications (OOPSLA'97), *Workshop on Object-oriented Behavioral Semantics (with an Emphasis on Semantics of Large OO Business Specifications)*, Atlanta, USA, 6th October 1997; Editors Haim Kilov, Bernhard Rumpe and Ian Simmonds; Proceedings of Munich University of Technology, TUM-I9737, September 1997