Formalizing UML; Mission Impossible?

Position paper at OOPSLA'98 workshop #9; Formalizing UML; Why?, How?

Peter van Emde Boas

ILLC, WINS, Universiteit van Amsterdam, Plantage Muidergracht 24, 1018 TV Amsterdam

Abstract

Formalizing UML will lead to problems much larger than those encountered in formalizing ordinary computer related languages. These problems concern not primarily the syntax. In fact that part is most likely to be formalizable at all. Instead I expect major problems with specifying a semantics as long as its purpose is undecided. Also the fact that UML is a hybrid of languages rather than a single one will make life difficult. Finally the fact that UML is visual yields unusual problems in the pragmatics of the language.

The Theoretician's Perspective on this Workshop; what is at Stake?

My backgrounds are in Mathematics, Logic and Computation Theory. If somebody refers to *UML* using the phrase "language" (as this is done by many people including many of you participating in our workshop), I expect that this language has the traditional properties, like having a *syntax* describing its structure and a *semantics* describing its meaning. And then should be such a thing as the *pragmatics* of the language, describing its rationality, use and purpose.

Attempts to formalize UML must address all these issues, possibly even at the same time. We better agree at any time during our discussions which issues are at stake at which time in whose contribution; otherwise confusion will be rampant and chaos will prevail.

The purpose of my contribution is to raise a few basic issues related to formalization as a research activity. My observations hardly concern UML itself; they apply to any project attempting to formalize something as big and complex as UML. In fact I know very little about UML. Worse, I have never been involved with the design or specification of any Information System at all. Yet it is my opinion that just reading a single introduction to UML like [2] will suffice for a computer scientist trained in logic and semantics, for finding reasons why it will be a hard, if not impossible job. Also, with my experience in working with formal systems, I can indicate which problems surface in in the workshop are core, and which are less important side issues, and indicate ways and means to look for solutions.

Existence of a Syntax and Semantics of a language are basic preconditions for the language to be formalizable at all. Syntax helps in determining whether something belongs to the language or not; Semantics is relevant for deciding whether two things in the language are equal or not. Both determinations must be possible in order that the entities in the language are real entities in some Universe of Discource, and I suppose that most of you would require that UML designs behave in this respect as proper entities.

Additional complications arise out of the fact that UML is not a single language but a hybrid of several languages. Which additional problems and challenges does this fact offer to a formalizer? Finally UML is a *visual* language. There are good reasons for being visual. However in our world most emphasis on formalization involves textual languages, or, if visual languages are dealt with at all, they are investigated by way of textual representations of the visual information. This state of affairs has new and unexpected pragmatic consequences.

In my contribution to this workshop I will present an abstract perspective on these four issues: formalizing *Syntax* and *Semantics* for a *Hybrid* and *Visual* language. The choice of these four topics is however not random; they all relate themes in logical semantics which have been or are being considered within our ILLC institute in Amsterdam.

Formalization is a tool which may assist in finding *truth*. Truth becomes important particularly in the context of *conflict*. Consequently, a guideline for our considerations is to look for the conflicts which may be answered by the formalization. I will illustrate my four topics in the context of trivial and in fact rather ridiculous scenarios, hoping to get some perspective on the problems involved.

Formalizing at the level of Syntax is, in principle, possible

Chris is presenting to you his latest design, constructed in UML. Looking at the funny symbols and nice colors you are wondering whether by accident he has exchanged his design with a drawing produced the day before by his eight year old daughter...

Determining whether a given expression belongs to a language or not is the prime purpose for a *syntax* of the language in question. It is in general the very first step in the construction of a formalization of the language. Ideally the syntax should provide you with a fast and efficient test for the validity of an expression. Does the given expression belong to the language or not. A weaker requirement is that the test exists but isn't always efficient. An even weaker requirement is that the syntax provides no test but a procedure which will generate all legal expressions and never an illegal one.

Contrary to what you would expect in a world were compilers for computer languages are generally available, most computer related languages which have actually been formalized by providing a syntax satisfies the weakest requirement only.[1] This holds particularly for languages where the correctness of some expression entails some semantic property. Still, having such an *Generative Grammar* suffices for calling the language formal.

In order to show that UML is formal in this sense one would expect that one needs first to select an appropriate format for a generative grammar and next an explicit construction of a generative grammar for UML (which would amount to completing a substantial part of a formalization). However this intuition is incorrect: formalizability of UML can be argued for on different grounds as well, for example, on the basis of pragmatic arguments only: *Pragmatics entails Syntax*.

Consider the following four simple assumptions:

Assumption 1 *Some object π is a UML design if and only if it describes some system p.*

Assumption 2 *A design π describes some system p if and only if there exists a witness certificate w establishing that the design describes the system*

Assumption 3 *Given an candidate certificate w, a system p and a candidate design π it can be decided whether w witnesses the fact that π describes p.*

and, given the fact that we are working over the internet:

Assumption 4 *The message expressing that the argument w witnesses the fact that the design π describes system p, can be communicated over the Internet.*

These four assumptions suffice for showing that UML is a formal language in the sense of having a generative syntax.[2]

The consequence of this by itself rather trivial observation is, that, before attempts are being made to construct the actual generative framework for an grammar for UML, one better decides what one expects this framework to do. In particular, the more one intends to enforce the *correctness* of a UML-design by means of the grammar, the harder the construction of such a grammar will be.

[1] such languages are called *recursively enumerable languages*.

[2] Participants attending last years' OOPSLA at Atlanta may recognize this argument. Last year I used it to establish that Design Patterns populate a formal language. [4]

As long as the *WHY*-question is unanswered, the *HOW*-question is meaningless.

George, System Development manager at Goodies Industry Inc. is having a meeting with his senior design staff. Alice, and Bob, the leaders of two design teams both have produced a design for the new Y2K-secured Information System, evidently in UML. Alice just has given a half hour argument that both designs in fact are the same. Bob has argued that the designs are different. How must George decide who is right?

There are trivial ways of answering this question which evidently disregard the issue at stake. Alice and Bob have used different sheets of paper to draw their design on, hence, as physical objects, they are unequal, and Bob is right. George might look at the electronic files inside the computer, and compare the two. In the unlikely case that these files turn out to be equal Alice evidently is right, and nobody will understand that Bob even could make a case.

A slightly more interesting case: the drawings could consist of similar visual elements, combined by similar connectors, only some of the identifiers used as labels and the positioning of the graphical objects in the plane are different.[3] My guess is that most of you will agree with Alice. But will this remain so if the conflict is moved to court, when Bob has left the firm and is sued for breaching his non-disclosure contract? What if Bob has made some simple but evidently reversible modifications to the Object-Types diagram.

Formalizing a language by designing a syntax and a semantics automatically will lead to a notion of *equivalence* among syntactically correct expressions in the language: two expressions are equivalent if they have the same *meaning*. In the above conflict it is not the identity of the designs themselves which is at stake but the identity of their meanings. What meaning is intended in fact may be depend on the context as illustrated below. The context may provide indications on the *requirements* on meaning, which may guide a formalizer towards some answer. These requirements relate the *WHY*-question more than the *HOW*-question.

UML being a hybrid language generates new problems.

Dave was collecting the sheets of his design when Lucy opened the window of the office. The pile of paper was scattered all over the floor. Dave had forgotten to put page numbers on his sheets. Picking up a single sheet he wondered which diagram in his design this sheet belonged to?

UML is not a single language but a series of related languages describing a variety of diagrams for various purposes. Yet the diagrams may contain similar visual ingredients which occasionally even denote the same entities. The designers of UML intend that you can navigate through the various diagrams in a design following the various objects, classes etc. Does this require these entities to preserve their meaning when going from one diagram to the next one? Can one accept the state of affairs where two expressions have equal of different meanings, depending on the sublanguage in which these expressions are being considered?

That there is a problem here can be recognized by "following" a single notion through the different diagrams: the notion of truth itself. Where Class and Object diagrams are living close to the Universe of discourse, which after all is a mathematical model of some part of the world itself, a diagram like the deployment diagram is modeling the available computers in your company. Truth with respect to a mathematical model is something we know how to formalize. Truth with respect to the real world is a problem which Philosophers have failed to resolve over a period of 4000 years. We can't expect these problems to be solved during the course of an experiment in formalization which in essence is the outgrow of a commercial standardization attempt.

The traditional approach in formalizing the semantics of a very complex language is to isolate the core concepts of the language under consideration, and embedding these core notions in a simplified *toy language* which includes just enough ingredients of the original language to make

[3] the typical situation encountered by the teacher of a programming course who suspects that one of his students has copied the solution of another.

the semantic issue under consideration meaningful and manageable. For UML this would mean constructing a semantics for each of the diagram types in isolation (or even just for sublanguages thereof). There exists an extensive literature on Abstract Data Types, Encapsulation, Inheritance mechanisms and Concurrency where such investigations have been performed with satisfactory results. However, such a treatment makes it hard if not impossible to perform the tracing of meaningful entities through all the diagrams. In OO the interaction between all these reasonably well understood semantic areas seems to be the main problem. What one needs is a formalism which looks at the hybrid structure of a language as a research topic in itself.

The good news is that there exists a rather recent formalism in logic which makes it possible to obtain hybrid semantics for hybrid languages. The tool is called *fibred semantics* [3]. The main idea is to consider the composed expressions of one language to be just atomic expressions in another. When evaluating the semantics you should switch your interpretation framework upon hitting such an atomic expression, which is composed in another language.

Evidently I cannot promise that this particular approach will help solving the problems of formalizing UML. But it might be a first step with assistance from an unexpected source.

UML being a visual language generates new problems.

Elvira was quite surprised to see that her programming team had grouped together a collection of unrelated classes in a single component. Upon enquiring she was told that on her diagrams these classes all had appeared together in the left upper corner of her drawing...

The story goes that battles about the visual appearance of objects in diagrams formed one of the main issues during the method war. Individual objects are just a small part of a visual language, and in fact the real issues should be how to combine smaller visual ingredients into larger ones in a well defined and meaningful way. You would want the entire formalization (both syntax and semantics) to reflect this two dimensionality of the language, and to diminish the dependence on the additional use of textual information in the labels.

In the good old world of string languages we have exploited for almost forty years the results from the theory of formal languages. This theory provides us with characterizations of language classes both at the level of generative procedures and at he level of recognizers; moreover the theory shows that both mechanisms have appropriate versions characterizing the same language classes. Attempts to obtain similar results for the world of visual languages have resulted in far less satisfactory characterizations. The best results have been obtained for grammar classes describing the topological and/or combinatorial structures which mathematicians call *graphs*. Consequently these results are relevant for the formalization of UML, since after all all diagrams in UML can be recognized to be instances of some sort of labeled graphs. Still simple notions like the result of application of a grammar rule or the influence of context turn out to be far more complex in two dimensions than in one dimensional world.

There is however another issue at stake which becomes relevant if we consider the third ingredient of a formalization: the *pragmatics* of a language. Pragmatics involves the study of a language in its use for the purpose it was designed for. Diagrams in UML will be used by system designers for communicating their ideas. The choice of going two dimensional was justified by the principle that a drawing can explain some idea much faster than a thousand words. Are our two dimensional grammar structures living up to this promise or not?

Let us assume for the moment that we have solved all *conceptual* problems about syntax and semantics. Our diagrams consist of fully standardized visual ingredients, connected using standardized lines, labels and other connectors. Moreover we have agreement on the formal meaning which is expressed by these diagrams. Does this means that the recipient of a design will always find in these diagrams what the author thinks he has put down there?

What is left unmentioned above is the problem of distributing the individual graphical ingredients over the available space. The problem of automated construction of drawings of graphs given by a combinatorial description has become an independent topic in computer science: *Graph*

Drawing. The algorithms investigated in this field can ensure that undesired crossings in graphs are prevented or minimized and that the drawings don't become too crowded. However, what has been drawn by one designer must be observed by another, and this observation is performed by means of the human visual system. And, as is well known among researchers in image processing and robotics, it so happens that the human visual system is one of the least understood and hardest to automate problems in computer science and technology.

Elvira above is evidently a victim of the *Gestalt* phenomenon, which makes observers group visual objects together on the basis both of proximity and of observed regularity. In the field of data visualization and the design of diagrams this phenomenon gradually is being understood, leading to guidelines for removing such *false implicatures*[4] from graphics [1]. Since the purpose of a language like UML will be to exchange information among humans rather than between humans and computers, these Gestalt phenomena will have to be taken into consideration also. Evidently this problem will resurface as an ingredient in a formalization of the pragmatics of UML.

References

[1] Dastani. M. M., *Language of Perception*, ph.d. thesis, University of Amsterdam, Dec 09 1998.

[2] Erikson, Hans-Erik & Penker, Magnus, *UML Toolkit*, J. Wiley, 1998.

[3] Gabbay, Dov M., *Fibring Logics*, Oxford Univ. Press, 1998.

[4] van Emde Boas, P. *Resistance is Futile; Formal linguistic observations on design patterns*, rep. ILLC-CT-97-02, Feb 1997; see: http://www.wins.uva.nl/research/illc/ResearchReports/CT-97-02.text.ps.gz

[4]patterns which seem to convey semantic meaning in a drawing which is not intended by the designer.