Improving the Definition of UML

Greg O'Keefe

Research School of Information Science and Engineering, Australian National University, Canberra, ACT 0200, Australia. greg.okeefe@anu.edu.au

Abstract. The literature on formal semantics for UML is huge and growing rapidly. Most contributions open with a brief remark motivating the work, then quickly move on to the technical detail. How do we decide whether more rigorous semantics are needed? Do we currently have an adequate definition of the syntax? How do we evaluate proposals to improve the definition? We provide criteria by which these and other questions can be answered. The growing role of UML is examined. We compare formal language definition techniques with those currently used in the definition of UML. We study this definition for both its content and form, and conclude that improvements are required. Finally, we briefly survey the UML formalisation literature, applying our criteria to determine which of the existing approaches show the most potential.

Many would argue that UML has no semantics [HR04,HS05], despite the numerous subheadings with that title in the documents which define the language [Obj06,Obj03,Obj05c,Obj05a]. Bran Selic [Sel04] counters these claims by collecting and summarising the scattered material on semantics from the main official document [Obj05c]. He also encourages theoreticians to study ways of making the semantics more precise.

The only real disagreement here is over the usage of the word "semantics." This is the topic of Harel and Rumpe's excellent article [HR04], and their position is that "semantics" is a mathematical term:

> Regardless of the exposition's degree of formality, the semantic mapping $M : L \longrightarrow S$ must be a rigorously defined function from the language's syntax L to its semantic domain S. Needless to say, an adequate semantic mapping for the full UML does not exist.

Selic, we believe, takes "semantics" to be an ordinary English word. Calling the prose from the official UML documents "semantics," is just saying that it describes the intended meaning of the models. The official UML documents exhibit an appreciation of the distinction between ordinary and technical usages:

> It is important to note that the current description is not a completely formal specification of the language because to do so would have added significant complexity without clear benefit.
>
> The structure of the language is nevertheless given a precise specification, which is required for tool interoperability. The detailed semantics

> are described using natural language, although in a precise way so they can easily be understood. Currently, the semantics are not considered essential for the development of tools; however, this will probably change in the future. [Obj03, §8]

This quote sets the scene for our investigation. It notes that some degree of precision is required to fulfil UML's mission. It claims, with a little hesitation, that this has been achieved without the use of rigorous mathematics. We will argue that there is a need for improvements, which we will identify.

The task is not to invent a new language, but to improve the definition of an existing one. The building industry has analogous situations. Sometimes a building of cultural significance is found to be structurally lacking. The builders will often suggest bulldozing it, and starting afresh, or making insensitive modifications like replacing a timber floor with concrete.

Too much of the UML formalisation literature takes the ham-fisted builder's approach to the problem, largely ignoring the existing definition, omitting large parts of the language or suggesting significant changes to it. We propose instead a minimal and sensitive adaptation of the existing definition to make it strong and stable enough, and more suitable to its new usage in model driven development. Like a good restoration architect, we should carefully consider the option of leaving things as they are.

Throughout this paper, we state criteria by which UML definitions ought to be evaluated. The first overarching criterion captures the conclusion just reached.

Criterion 0 *An improved definition of UML should not change the language or the definition any more than is needed to enable UML to fulfil its role.*

In our first section we consider the task of defining a language, and in the second, we show why the semantic part of the definition is important. The third section examines the purpose of UML, and in the fourth we study two of the more difficult aspects of the current UML definition. The fifth section evaluates the existing definition of UML and the sixth briefly surveys the literature to identify the most promising efforts to improve that definition. We conclude by saying how we hope the future of UML semantics research will differ from its past.

1 Defining Languages

Diagrams do not need to conform to some defined language in order to help us communicate. People find it quite natural to express their ideas by drawing pictures, as any survey of publications, presentation slides or white-boards will verify. Most of these diagrams do not conform to any specified diagram type. If they are part of a language, it is a *natural* language, like English[1].

[1] We will speak of "English" when we mean any arbitrary natural language such as English, Occitan or Brazilian Portuguese.

A description of a natural language is a scientific theory, which must be judged by how well it predicts actual usage. Artificial languages on the other hand, are *defined*. Usage which does not conform to the definition is incorrect.

Sometimes the primary purpose of creating diagrams is not to communicate ideas, but rather to generate or organise them. The “mind maps” technique [Buz95] is one example. UML can be used in this way too. Building a UML model can drive the collection of information about a problem domain, and provide a convenient structure for organising that information. This role does not, however, conflict with its status as a defined language.

The mind-map book provides guidelines for creating and reading these mind-maps, which we might, very charitably, regard as a language definition. It is certainly not a precise definition, nor is it intended to be, because precision simply is not required. In a commentary attached to the amusing article “Death by UML Fever” [Bel04], Philippe Kruchten implies that UML does not need to be precisely defined.

> UML is a notation that should be used in most cases simply to illustrate your design and to serve as as a general road-map for the corresponding implementation.

UML can be used as documentation of code, but it is also intended as a means of *specifying* a system. Model Driven Architecture (MDA) [MM03] calls for complete systems to be generated automatically from UML models. If the language is not precisely defined, the generated system may not be what the model creators intended.

Computer programs are usually written as linear text, but compilers and interpreters parse this text into a tree-like structure which is easier to process. These structures are called the abstract syntax. Similarly, UML has an abstract syntax which is processed by model transformation and code generation. The relation in UML between concrete diagrammatic syntax and the abstract syntax it represents, is complicated enough to be a potential source of error. Precisely defining this relationship could simplify the creation of graphical model editors, and facilitate animations [EHHS00, §6] and reverse engineering. The definition should clearly delineate concrete syntax, abstract syntax and semantics, and it should also specify the relationships between these parts. We therefore require that

Criterion 1 *A UML definition should unambiguously define*

concrete syntax *the diagrams and other notation*
abstract syntax *the UML models*
notational conventions *a unique model for each diagram collection*
semantic domain *the abstract systems which models “talk about”*
semantics *whether a given model is true of a given system*

2 Applied Semantics

Avoiding possible disagreements about whether or not a given system satisfies a model is enough to motivate the semantic parts of Criterion 1. Model driven development raises other questions whose answer depends on well defined semantics.

Since the abstract syntax of UML is defined by a UML metamodel, we actually require a subset of the semantics to even know whether an alleged model actually is a well-formed model.

We need it to be clear whether or not a given model is consistent. That is, can some system satisfy this model? When we have separately modelled distinct aspects of an envisaged system, we need a system which satisfies all of the aspect models. Model consistency includes: preservation of association multiplicities and other invariants; satisfaction of pre-post-condition contracts by object behaviours; satisfaction of use-case contracts by a model; safety properties (bad things can not happen) and liveness properties (system does not get stuck).

If a model is made more concrete as a project progresses, we may wish to determine whether the more concrete model is a refinement of the more abstract one. Indeed, we may wish to establish once and for all that a certain model transformation always produces a refinement of its input model. We tentatively call such a model transformation *sound*. Refinement and soundness have various mathematical definitions, but this is not the place to make these choices. Note however that it is not enough to say that one model is a refinement when it adds some detail, because we probably want to consider non-trivial model transformations like the famous class to database schema example [BRST05] to be a kind of refinement.

We not only want these questions to have definite answers, but we would also appreciate any tool support in finding these answers.

Criterion 2 *A UML definition should settle the following questions:*

model consistency *is there a system which satisfies all these models?*
model refinement *is this model a refinement of that one?*
transformation soundness *does output model always refine input?*

The definition should also support maximally automatic tools to help determine the answers to these questions.

3 Working with Ideas

Bran Selic has wisely observed that "software development consists primarily of expressing ideas" [Sel03]. A project attempts to improve some situation by introducing or modifying a system[2]. Ideas describing the situation must be expressed, absorbed, discussed, analysed, tested, revised and agreed on. The system itself must also be described, both at a high level, in terms of the ideas about the

[2] I am indebted to Shayne Flint for this view of engineering

situation, and at the low level, using ideas about specific technologies. The high and low levels must agree, and all the ideas must be clear and free from confusion and contradiction. Indeed, the part of software development that is not about expressing ideas is mostly about generating, negotiating and translating them.

High level languages have contributed enormously to development productivity [Bro87], but ideas expressed in Fortran or Java are still far from the requirements level ideas of the human beings for whom a system is built. It is well known that requirements are not fully known, understood or agreed on at the beginning of a project, and that they will change before project completion. Hence effective software development requires the most direct possible coupling between the thoughts of the stakeholders and their expression in implementation languages.

When a skilled programmer writes code for her own purposes, this coupling is perfect. The jewels of computer programming are usually formed in this way. Extreme programming and other agile processes seek to couple high and low level ideas by constant face-to-face communication between stakeholders and programmers, and frequent delivery of useful code to stimulate feedback. These forms of idea coupling depend heavily on individuals. For large projects and organisations, it is desirable for the coupling of ideas to be systemic. This can be achieved by establishing model transformation and code generation chains.

We agree with Steve Cook that "...for a language to be usable to drive an automated development process, it is essential for the meaning of the language to be precise" [HS05]. Without an agreed precise meaning, an automatic translators interpretation of a model might differ from that of the stakeholders. Then the delivered system might be unsatisfactory, even dangerous. The definition of UML should therefore provide a reference for those who build model translators.

Criterion 3 *UML and friends should enable people to reach agreement on, and to directly express ideas about:*

problem domains *telecommunications, finance, logistics, ...*
implementation platforms *linux cluster, enterprise Java, ...*
translation *between these representations*

The definition should enable tools to agree with people about what these expressions mean.

We prefer to speak of "direct expression" rather than "raising the level of abstraction." A highly abstract expression of ideas might still be far from the stakeholders understanding, and thus not particularly useful.

A widespread agreement between users and toolmakers about the meaning of UML would enable trade in models and transformations. This would in turn greatly reduce the cost of developing systems. Brooks [Bro87] notes that the ability to buy software rather than build it has contributed greatly to reducing software cost. Organisations whose needs can not be met by direct purchase of software might one day be able to purchase models and transformations which can be assembled to satisfy those requirements much more cheaply than "ground up" development.

4 The Definition of UML 2.0

UML 2.0, we have observed, is not defined in the way artificial language experts normally do business. How then is it defined? In this section we will take a brief look at the small mountain of documentation [Obj06,Obj03,Obj05c,Obj05a] which defines UML. These documents will be collectively referred to from here on as *the definition*.

4.1 Metamodelling, Metacircularity and Reflection

The long and complicated story that is UML's definition begins with the "Infrastructure Specification" [Obj03]. This gives a UML model called the "Infrastructure Library," which "contains all the metaclasses required to define itself" [Obj03, §7.2.8]. The Meta Object Facility (MOF) [Obj06] builds on the infrastructure library to create a metamodelling language used to define UML [Obj05c]. This definition of UML proper begins by including the infrastructure library.

This technique, of using a modelling language to define a modelling language is called *metamodelling*.

Metamodelling need not be circular. A metamodelling language with an independent definition can properly define the abstract syntax of a modelling language. The UML definition describes its usage of metamodelling as metacircular [Obj03, §8.1], because it uses a UML subset to define UML. Without an independent definition of the metamodelling language though, the "meta" seems like an unwarranted euphemism.

Because the metamodelling language used to define the abstract syntax of UML is a subset of UML, that abstract syntax inhabits the semantic domain of the language. Having the syntax inside the semantics is also required in order to make sense of one of UML's notions of instantiation. Consider a model with a class C and an instance specification $: C$. Although it would be redundant in this situation, we join the instance specification to the class with an ≪`instanceOf`≫ arrow. Fix a semantic mapping i (interpretation) which takes the instance specification to an instance, and the class to a set of instances. The situation then can be depicted as shown in Figure 1.

$$\begin{tikzcd}
\boxed{:\mathrm{C}} \arrow[r, "\ll\texttt{instanceOf}\gg"] \arrow[d, "i"'] & \boxed{\mathrm{C}} \arrow[d, "i"] \\
c \arrow[r, "\in"'] \arrow[ur, "instanceOfType"] & C
\end{tikzcd}$$

Fig. 1. Semantics of Instantiation

Ignoring the *instanceOfType* arrow for a moment, we have a neat separation between syntax on the top line, and semantics on the bottom. So we see that the ≪`instanceOf`≫ notation in a UML diagram corresponds to "element of" ($\in$) in the system state.

The operation *instanceOfType*, defined in MOF for the metaclass Element, "returns true if this element is an instance of the specified Class…" [Obj06, §13.3]. The arrow in Figure 1 marked with this name, indicates an Element, Class pair where the operation returns true. The operation crosses the syntax/semantics divide. To make sense of such reflective notions, we not only require the syntax to be in the semantic domain, we actually need each syntactic model to be present in every system state which satisfies it.

Element is a superclass of everything in UML, and of most things in MOF. However this *instanceOfType* operation is only present in the MOF version. The superstructure explicitly disowns such reflective ideas: "The [action] semantics are also left undefined in situations that require classes as values at runtime" [Obj05c, §11.1]. A distinction is sometimes drawn between "runtime semantics" and "repository semantics" [Obj05c, §6.3]. We do not consider it necessary or desirable to support two distinct semantic definitions for what is essentially the one language. It would add work, and potentially lose the benefits of tool reuse between metamodel and model levels. The differences arise because the metamodels, as we have just seen, are slightly different. We therefore require semantics that can account for reflective operations, even though the current definition chooses to ignore them at runtime.

We summarise our findings in the following criterion. Although the last two points entail their predecessors, we list them to provide a range of "compliance levels" (in the style of [Obj05c, §2.2]).

Criterion 4 *The definition of UML should satisfy*

unity *common semantics for repository and runtime*
self-containment *semantic domain contains abstract syntax*
reflection *model contained in each of its instances*

4.2 Varieties of Variation

The UML definition contains a great number of "semantic variation points." These are places where the semantics are explicitly undefined, or where a range of possibilities are allowed. Chapter 18 of [Obj05c] describes the profiles mechanism of UML, which allows subsets and extensions of UML to be defined. Model driven development may also call for domain specific languages which can interoperate with UML models. Finally, UML 2.0 is only the latest of many revisions of the language, and will not be the last. For all these reasons, we require semantics which are *flexible.*

Criterion 5 *The definition of UML must enable the language to be adapted and extended. In particular, it requires a "semantic envelope" [Sel04] which enables precise treatment of:*

semantic variation points
profiles
domain specific languages *interoperable with UML*
later versions *of UML*

5 The UML Definition Evaluated

Having established the properties that a definition of UML ought to have, we turn now to the existing definition and ask, is it any good? We begin with a perfect score on Criterion 0, since no definition can be more faithful to the current definition than the current definition.

Criterion 1 can be summarised by saying that any proposed "definition" of UML should actually define it. Debates on whether or not a given diagram is a correct UML diagram, whether a system satisfies a given model and so on, should be easily resolved by referring to the definition. Indeed, if the definition was clear and understandable, these debates would seldom occur. That is to say, satisfying Criterion 3 on enabling agreement, is probably our best indication of whether Criterion 1 has been met. We therefore consider Criterion 3 before returning to Criteria 1 and 2.

UML does not fulfil Criterion 3 so well as we could hope, because users are not currently able to easily reach agreement about the meaning of a model.

> ... many people are confused about what these [UML] concepts ... really mean and how to understand and use them [HR04]

> Developers can waste considerable time resolving disputes over usage and interpretation of notation. [BF98]

We have had similar experiences when attempting to extract the precise meaning of a diagram from groups of experienced UML practitioners: diverse interpretations each received vigorous support. Debate continues at the OMG over fundamental matters such as the semantics of associations and their ends [Obj, Issue #5977][Mil06]. It seems fair to conclude that there is not widespread agreement about the meaning of UML models.

It is not valid to infer from this that the definition lacks precision, because the lack of agreement could be the result of the definition being difficult to understand. This would be unfortunate, since it explicitly strives for understandability, even at the cost of some precision [Obj03, §8] (quoted on Page 1). To us, it seems more plausible that the definition is neither precise nor understandable.

The quote from the UML definition argues that a mathematical approach involves too much work, and is not necessary to get the job done. Whether or not Greek letters and other fancy symbols are employed, precise definitions of abstract ideas *are mathematical*. If we choose to ignore the accumulated wisdom of the mathematical discipline, and define things our own way, we commit the same error as "hackers" who refuse to follow established software engineering practice. Like the hackers, we are likely to get ourselves into the kind of trouble that the experts know how to avoid. One simply does not find disagreements about the meaning of definitions in mathematics, but after almost 10 years even the basics of UML are still in dispute.

Turning to Criterion 2, one could hardly hope to settle questions of model consistency, refinement and transformation soundness without true definitions of the relevant concepts. It should not surprise us then that Stephen Mellor finds a lack of support for model consistency testing in the current definition [HS05]. He claims that the definition fails to detect the apparent inconsistency of his small example model. We conclude then that the definition rates poorly on Criteria 1, 2 and 3.

Criterion 4, on reflection, is really a detail of Criterion 0, since it records what is entailed by the definition. Criterion 5 on flexibility, is only challenging for a rigorous definition. "Semantic variation points" offer perfect flexibility. Interpretation of the metamodel diagrams by object-oriented folk-law is sufficient for current tools, which only manipulate the syntax. Without adequate support for the other criteria though, these benefits are of little use.

To achieve a definition which satisfies our criteria then, we may have to tolerate a little mathematics. The next section surveys some of the work applicable to this task.

6 The UML Formalisation Literature

Since the current definition does not satisfy the requirements, we would like an improved definition for UML. The new definition should agree with the current one, including its reflective metamodelling approach, it should define the semantics sufficiently to enable automated checking of consistency, refinement and soundness, and it should be flexible and understandable. We now take a brief look at some work related to improving the definition of UML, in the light of our criteria.

Kim, Carrington and Burger [KC00,KBC05] give explicit translations between Object-Z and class diagrams. In the earlier work, the syntax of both languages is expressed in Object-Z, and the translation is also defined there. A metamodel of Object-Z is provided for the benefit of modellers unfamiliar with this formal language. In the later work, the metamodels define the syntax, and the translation is defined using a dedicated model transformation language. Unfortunately, even this recent work only addresses a subset of the class diagram fragment of UML. The work aims to enable formal verification of UML models, but as yet we have no demonstration nor descriptions of specific techniques.

Model Driven Architecture [MM03] aims to enable the simultaneous use of many languages, each with syntax defined in MOF, by using model transformations between these languages. The real contribution of [KBC05] is in recognising that formal languages can also participate in this way. Definitive formal semantics could be provided by a Z (or Object-Z or dynamic logic or . . .) metamodel and UML to Z model transformation. This would enable tool integration, and provide insight into the formalism for the more advanced modellers. Attempts to directly translate diagrams into formal languages usually ignore the metamodel definition of the language, and thus violate Criterion 0.

Bruel and France [BF98] advocate an integration of UML and formal methods, in which a UML class diagram is translated into the formal specification language Z. The Z specification is then manually refined, adding details not expressible using class diagrams. The rules and guidelines for semi-automatic translation, they hope, will give insights for developing a more precise semantics for UML.

Rasch and Wehrheim [RW03] also advocate integration of a formal language, in this case Object-Z, into the development process. The Object-Z specification manually derived from the class diagram also specifies the class operations. The class is further constrained by a protocol state-machine, which together with the Object-Z schema, is translated into CSP. The choice of CSP, which is even less readable than Z, seems to be motivated mostly by the availability of a model checker[3] which they aim to use for consistency testing. They consider several notions of consistency and study which of these are preserved under CSP notions of refinement. We are not convinced that the intended semantics of the UML fragment are captured by this translation. It is also not clear that the CSP notions of refinement are applicable. We see little hope that modellers and transformation authors will become familiar with both Object-Z and CSP.

The association end annotation {`unique`} is the subject of a recent controversy [Obj, Issue #5977]. Dragan Milicev [Mil06] proposes semantics which reconcile the apparently conflicting parts of the UML definition. These semantics concern associations, their ends and the read, create, and destroy link actions. In an appendix to the report, Milicev gives an example model to illustrate the controversy, and expresses his semantics for it in Z. This is intended merely as a precise statement of the proposal explained in the body of the paper. However, this is the most convincing example we have seen of using Z to express dynamic aspects of UML. It is also a good example of why Z will never be widely used by developers: it is not easy to read.

Algebraic specification extended with "generalised labelled transition systems" is used by Gianna Reggio, Maura Cerioli and Egidio Astesiano to formalise parts of UML in [RCA01] and earlier papers by the same authors. They do this by translating UML diagrams into the language Casl-LTL, though they emphasise that the particular language is immaterial. This work explicitly aims for a way of giving useful formal semantics to the whole of UML, and as the title suggests, they take seriously the idea that the different diagrams combine to specify a single system. However they ignore the fact that the official definition already interprets the variety of diagrams into a single abstract syntactic entity, the model. The authors note the expressive demands made by UML's dynamic diagrams.

> It is worth noting that to state the behavioural axioms we need some temporal logic combinators available in Casl-Ltl that we have no space

[3] Note that this is not a tool intended for checking UML models. "Model" here is a technical term from symbolic logic, meaning interpretation.

> to illustrate here. The expressive power of such temporal logic will also be crucial for the translation of sequence diagrams...

Indeed, Z and its derivatives would face similar difficulties. A later paper [AR02] by Astesiano and Reggio studies UML consistency from their algebraic point of view, and also uses a metamodel to describe the formal language being used.

The Object Constraint Language (OCL) [Obj05a] is very much like the languages of traditional symbolic logic, and at least two groups have attempted to make it precise by translating it into well understood systems of logic, intending to enable theorem proving about models. Brucker and Wolff [BW02] use higher order logic (HOL) as implemented in the generic interactive theorem prover Isabelle. Beckert, Keller and Schmitt [BKS02] use first order logic. OCL 2.0 has a third truth value "undefined" and allows collections of collections, so first order logic will probably not suffice to formally define it. Neither group make use of the OCL metamodel in their translations. Beckert's group offer different, equivalent translations optimised for readability or for automated theorem proving respectively. With a foundation as suggested in these works, OCL itself could be the target formal language for a model transformation defining the semantics of UML. This would probably require additions to the current limited temporal operators of OCL though.

The OCL formalisation of Beckert and Schmitt [BKS02] is used in their the KeY project [ABB$^+$05]. This is a tool for the deductive verification of Java-Card programs using a specialised dynamic logic [Bec01]. This logic is implemented in a generic theorem prover integrated with the Together modelling tool, and thus provides a practical platform integrating UML modelling and formal methods. Although this work is not aimed at improving the definition of UML, it is instructive. The deductive rules symbolically execute the Java-Card program, and thus give a clear and precise account of the language semantics. The rules could even provide educational interactive animations of the language.

Unlike Java-Card, UML is non-deterministic and has no `main` procedure, but it is conceivable that one could develop such a dynamic logic for UML. The logic would have rules for each of the UML actions. This would define model dynamics, and the meaning of each of the diagrams could be expressed by translation into the dynamic logic language. It would also enable deductive verification of UML models. In its traditional form, dynamic logic is even less readable than Z. But a UML specific logic could use OCL notation for its static parts, whilst the program parts would be written using the yet to be fixed standard UML action language.

Wieringa and Broerson [WB97] use a formal language derived from dynamic logic to give formal semantics for parts of UML class and state machine diagrams. As in earlier work by Wieringa, a "methodological analysis" leads the authors to diverge radically from the official definition: a system is a black box, which responds instantly to external stimuli. It is not possible for example to make sense of a sequence diagrams in such a system. This might be a useful interpretation of UML for requirements engineering, as these authors see it, but from our perspective, it is inventing a new language rather than providing a better definition of the existing one.

We take this opportunity to mention our own work [O'K06], which uses standard dynamic logic to give precise semantics to a UML subset with class, state machine and sequence diagrams, and send and receive actions. Tableau theorem proving techniques are employed to test model consistency. Other work on sequence diagrams require every occurrence to be made explicit in the diagram, whereas our formalisation allows hidden occurrences in between the explicit ones. This raises the level of abstraction, appropriately ignoring details that the sequence diagram author does not care about. This work makes no attempt to handle visibility and polymorphism issues. Standard dynamic logic is probably not suitable for a full UML fomalisation, as it lacks higher order expressions and parallel composition.

The first plausible demonstration of deductive verification of UML models is given by Arons, Hooman, Kugler, Pnueli and van der Zwaag, in [TAKP$^+$04]. The semantics are not described in this paper, but are derived from those of [DJPV03]. That paper gives formal semantics to a small executable subset of UML intended for real-time applications, using Pnueli's "symbolic transition systems." Much of the considerable complexity of that work comes from the need to model hard real-time systems, which makes us wonder whether the general modelling community might get by with something simpler. The abstract syntax of the official definition is ignored, and a traditional formal syntax is given for the selected UML subset. The later deductive verification work uses a temporal logic embedded into the higher order logic of the PVS interactive theorem prover. A model given by a class diagram and state machine diagrams with some actions, is automatically translated from `.xmi` form into PVS sources. Issues of consistency are deliberately avoided, since deductive verification of liveness properties and safety properties are challenging enough at this stage. Several strong assumptions are made about the execution semantics, which are not present in the official definition. Deductive verification is not required for most applications of UML, but supporting formal proof demonstrates that a definition is precise and unambiguous, which we have demanded in Criterion 1. This formalisation uses a language with both temporal and higher order features, so it is not subject to many of the limitations we have identified for other approaches. Most of our criteria are not addressed by this work however. Most urgently, we need techniques to check consistency, and we need the meaning of models to be understood by non-technical modellers and end users.

Several workers [ZHG05,EHHS00] employ graphs and graph transformations [BH02] to give formal semantics to UML. In this way, a system state can literally be an object diagram, which is clearly much easier to understand than the usual logico-mathematical offerings. The graph transformation rules, which define the system dynamics, can be given using UML collaboration diagrams [EHHS00]. Metamodels are usually static, consisting of only class diagrams. If we included collaboration diagrams in the metamodelling language, we could view metamodels as specifications of graph transformation systems. Thus we can define semantics for a modelling language by providing a model transformation from the modelling language to the metamodelling language! This would be a significant change to MOF, but seems well motivated and could win support. Unfortunately the specific metamodel proposed in [EHHS00] depends on the Object metaclass in 1.x UML metamodels. This is widely accepted to have been confused, and has been removed in UML 2.0. Model consistency from a graphical point of view is considered in [EHHS02]. Graph transformation offers the attractive prospect of a common language for practical software engineers and academic theoreticians. This combination of rigour and understandability, we believe, is the key to satisfying all our Criteria. The present author will be reading more about graph transformation.

All the work we have discussed takes part of UML and translates it into another language with formal semantics. An alternative would be to use English and elementary mathematics to define the semantics[4]. This approach would allow us to use specific formalisms for specific tasks, whilst avoiding their expressive limitations when defining the semantics. It is easy to adapt an English/mathematical text, but this does not automatically integrate the new interpretation with existing tools. An alternative is to directly define semantics for a core of UML, then translate the remainder of the language into this by model transformation. This seems to be the intention of the OMG's current request for proposal on an executable UML foundation [Obj05b].

Model transformation from UML to a language with precise semantics seems the most promising method for improving the definition of UML. The formal language must be able to express temporal and higher order concepts, handle scoping and polymorphism, and admit automated consistency checking. Perhaps the most challenging requirement though, is that it should enable people to better understand UML models.

7 Conclusion

Too much of the work on UML semantics looks like a technical answer which is glad to have found a good practical question. We have asked what that question actually is, and refined it in the form of criteria for an improved definition. It is our hope that future work will explicitly address the larger task of improving the definition of UML. Our criteria might serve as goalposts for formalisation work, or as targets for demolition by more worthy replacements. Either way, it is a step towards the more desirable situation where a practical question seeks a technical answer.

References

ABB$^+$05. Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Richard Bubel, Martin Giese, Reiner Hähnle, Wolfram Menzel, Wojciech Mostowski, Andreas

[4] This apparently obvious idea had not occurred to us before it was pointed out by Peter Schmitt.

Roth, Steffen Schlager, and Peter H. Schmitt. The KeY tool. *Software and System Modeling*, 4(1):32–54, 2005.

AR02. Egidio Astesiano and Gianna Reggio. An attempt at analysing the consistency problems in the UML from a classical algebraic viewpoint. In *WADT*, pages 56–81, 2002.

Bec01. Bernhard Beckert. A dynamic logic for the formal verification of java card programs. In *Java on Smart Cards: Programming and Security*, number 2041 in LNCS, pages 6–24. Springer, 2001.

Bel04. Alex E. Bell. Death by UML fever. *Queue*, 2(1):72–80, 2004.

BF98. Jean-Michel Bruel and Robert B. France. Transforming UML models to formal specifications. In *Proceedings of the OOPSLA'98 Workshop on Formalising UML*, 1998.

BH02. Luciano Baresi and Reiko Heckel. Tutorial introduction to graph transformation: A software engineering perspective. In *Proceedings of the first International Workshop on Theory and Application of Graph Transformation*, pages 402–429, 2002.

BKS02. Bernhard Beckert, Uwe Keller, and Peter H. Schmitt. Translating the object constraint language into first-order predicate logic. In *Proceedings of VERIFY, Workshop at Federated Logic conferences (FLoC)*, 2002.

Bro87. Frederick P. Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *Computer*, May 1987.

BRST05. Jean Bézivin, Bernhard Rumpe, Andy Schür, and Laurence Tratt. Model transformations in practice workshop, call for papers. web, July 2005. `http://sosym.dcs.kcl.ac.uk/events/mtip05/long_cfp.pdf`.

Buz95. Tony Buzan. *The Mind-Map Book*. BBC Books, 2nd edition, 1995.

BW02. Achim D. Brucker and Burkhart Wolff. A proposal for a formal OCL semantics in Isabelle/HOL. In V.A Carren no, C. Mu noz, and S. Tahar, editors, *TPHOLS 2002*, volume 2410 of *LNCS*, pages 99–114. Springer-Verlag, 2002.

DJPV03. Werner Damm, Bernhard Josko, Amir Pnueli, and Angelika Votintseva. Understanding UML: A formal semantics of concurrency and communication in real-time UML. In *Formal Methods for Components and Objects, Proceedings 2002*, volume 2852 of *LNCS*. Springer, 2003.

EHHS00. Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioural diagrams in UML. In *Proceedings of UML*, volume 1939. LNCS, 2000.

EHHS02. Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Testing the consistency of dynamic uml diagrams. *Integrated Design and Process Technology*, 2002.

HR04. David Harel and Bernhard Rumpe. Meaningful modeling: What's the semantics of "semantics"? *Computer*, pages 64–72, October 2004.

HS05. Brian Henderson-Sellers. UML - the good, the bad or the ugly? perspectives from a panel of experts. *Software and System Modeling*, 4(1):4–13, 2005.

KBC05. Soon-Kyeong Kim, Damian Burger, and David A. Carrington. An MDA approach towards integrating formal and informal modeling languages. In *FM*, pages 448–464, 2005.

KC00. Soon-Kyeong Kim and David A. Carrington. A formal mapping between UML models and object-Z specifications. In *Proceedings of the First In-*

ternational Conference of B and Z Users on Formal Specification and Development in Z and B, pages 2–21, 2000.

Mil06. D. Milicev. On the semantics of associations and association ends in UML. Technical report, University of Belgrade, School of Electrical Engineering, February 2006.

MM03. Joaquin Miller and Jishnu Mukerji. MDA guide. Technical report, Object Management Group, 2003. `http://www.omg.org/mda`.

Obj. Object Management Group. Issues for the UML 2 revision task force. web. `http://www.omg.org/issues/uml2-rtf.html`.

Obj03. Object Management Group. UML 2.0 infrastructure specification. Technical report, Object Management Group, 2003. `http://www.omg.org/docs/ptc/03-09-15.pdf`.

Obj05a. Object Management Group. OCL 2.0 specification. Technical report, Object Management Group, 2005. `http://www.omg.org/docs/ptc/05-06-06.pdf`.

Obj05b. Object Management Group. Request for proposals: Semantics of a foundational subset for executable UML models, 2005. `http://www.omg.org/docs/ad/05-04-02.pdf`.

Obj05c. Object Management Group. Unified modeling language: Superstructure. Technical report, Object Management Group, 2005. `http://www.omg.org/docs/formal/05-07-04.pdf`.

Obj06. Object Management Group. Meta object facility (MOF) 2.0 core specification. Technical report, Object Management Group, 2006. `http://www.omg.org/docs/formal/06-01-01.pdf`.

O'K06. Greg O'Keefe. Dynamic logic for UML consistency. In *ECMDA-FA, European Conference on Model Driven Architecture*, number 4066 in Lecture Notes in Computer Science, pages 113–127. Springer, 2006. `http://rsise.anu.edu.au/~okeefe/dl4uml.pdf`.

RCA01. Gianna Reggio, Maura Cerioli, and Egidio Astesiano. Towards a rigourous semantics of UML supporting its multiview approach. In H. Hussmann, editor, *FASE 2001*, volume 2029 of *LNCS*, pages 171–186. Springer, 2001.

RW03. Holger Rasch and Heike Wehrheim. Checking consistency in uml diagramms: Classes and state machines. In *FMOODS*, pages 229–243, 2003.

Sel03. Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 2003.

Sel04. Bran V. Selic. On the semantic foundations of standard UML 2.0. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, number 3185 in LNCS, 2004.

TAKP$^+$04. J. Hooman T. Arons, H. Kugler, A. Pnueli, , and M. van der Zwaag. Deductive verification of UML models in tlpvs. In *Proceedings UML*, 2004.

WB97. Roel Wieringa and Jan Broerson. Minimal transition system semantics for lightweight class and behaviour diagrams. In Manfred Broy, Derek Coleman, Tom S. E. Maibaum, and Bernhard Rumpe, editors, *Proceedings PSMT'98 Workshop on Precise Semantics for Modeling Techniques.* Technische Universitaet Muenchen, TUM-I9803, April 1997.

ZHG05. Paul Ziemann, Karsten Hölscher, and Martin Gogolla. From UML models to graph transformation systems. *Electr. Notes Theor. Comput. Sci.*, 127(4):17–33, 2005.