Making UML Precise

A. Evans
Department of Computer Science
University of York, York, UK

J-M. Bruel
Laboratoire TASC
Pau University, France

R. France
Department of Computer Science
Colorado State University, Colorado, USA

K. Lano
Department of Computing
Imperial College, London, UK

B. Rumpe
Department of Computer Science
Munich University of Technology, Munich, Germany

http://www.cs.york.ac.uk/~ puml

December 8, 1998

1 Introduction

The Unified Modeling Language (UML) [BJR98] is a set of description techniques for specifying, visualising and documenting object-oriented (OO) systems. The language has been developed by Grady Booch, James Rumbaugh and Ivar Jacobson as a synthesis of their well known object-oriented methods and also combines a number of ideas from other methods and description techniques such as Harel Statecharts. UML is rapidly becoming an important industry standard for modelling software systems. Recently, UML version 1.1 has been accepted by the the Object Management Group as a standard notation for object-oriented analysis and design.

Like other software engineering 'methods' UML provides a set of graphical and textual modelling techniques that aim to be understandable to system developers and customers. Each technique is used to model the system from a number of different perspectives. For example, class diagrams are used to model static (data) properties; sequence diagrams are used to model flow of messages between objects. However, an important feature of UML is that it is a modelling language, not a method, and therefore does not prescribe any particular development process. Furthermore, UML provides a semantics documents which aims to precisely document the meaning and structure of the language.

It has long been recognised that many OO methods, including those from which UML is derived, suffer from a lack of a *precise semantics*. These methods only provide a very imprecise interpretation of the meaning of the diagrams they used. This can lead to a number of problems:

Interpretation: a lack of precise semantics can result in confusion and disagreement between the creator and reader of diagram over its precise meaning. Developers can waste considerable time resolving disputes over the meaning of a diagram and may walk away with different and conflicting views. If the system is being implemented by someone other than the creator of the diagram, they may implement an incorrect interpretation of the required design.

Design: a rigorous design process cannot be used unless there is a semantics upon which to develop refinement conditions.

Rigorous Analysis: without a precise semantics it is impossible to carry out a rigorous semantic analysis of a diagram. In practice, OO diagrams are validated and verified informally. These informal techniques are often inadequate; they cannot be used to rigorously establish that implementations and models are consistent with each other, or that models capturing different views of the system are consistent with each other.

Although UML provides a semantics document, it is clearly essential to determine whether it can resolve the problems outlined above. The precise UML group was formed in late 1997 with the aim of both inevstigating the completeness of the UML semantics and developing novel approaches to using UML more precisely. This paper aims to briefly present some of the issues its members have considered with regards to making UML precise and to describe progress that has been made over the last 12 months.

2 Formalization

One of the core issues considered by the group has naturally been that of formalization, i.e. should UML be formalized; is UML already formal enough and how should it be best formalized?

The desire to formalize UML was originally motivated by the overall wish to develop practical, industrial strength, formal methods. Members of the group had already worked with earlier OO methods in an effort to give them a sounder formal basis. The advent of the UML as a likely de-facto industry standard, and its recognition that as a standard it needs to be precisely described, made UML a natural choice for a combined investigation.

The original 1.0 version of the UML semantics did not provide a precise description of the language, nor did one of its predecessors UML 0.9, UML 0.8, OMT [RBP$^+$91], the Jacobson Method [Jac93], or the Booch Method [Boo94]. A meta-modelling approach was used to describe the language, in which class diagrams were used to capture the static relationship between modelling concepts. However, the well-formedness rules and semantics of the language were expressed in English rather than formal text. In version 1.1, a serious attempt has been to provide a more precise description of the language. Version 1.1 now supports the use of a semi-formal constraint language, the object constraint language (OCL), which is used to describe well formedness for the modelling abstractions supported by UML. Certain dynamic semantics properties of the model (e.g. the meaning of associations as object links) are also captured within the model.

How incomplete are the semantics of version 1.1 and indeed is meta-modelling a suitable approach to take in their definition? A (purist) answer to the latter question is that class diagrams and OCL are insufficiently precise to describe the semantics of a langauge. OCL itself does at present not have a formal semantics and a number of modelling abstractions used in UML class diagrams have still to be resolved. For example, the problems associated with aggregation have been extensively investigated in [SFLP98]. An opposing view is that that accessibility and compatibility of the language used to define the semantics should have precedance over formality. In this respect, class diagrams and OCL are an intuitive and intelligent choice of language as they will be understandable to users of UML.

With respect to the completeness of the semantics of version 1.1, it is clear that a significant amount of work is required in order to make them 'formally complete'. In particular, [Eva98b] identifies three core areas where the static semantics model of version 1.1 is incomplete: the meaning of inheritance, definitions of constraints on inheritance hierarchies and the meaning of aggregation and abstract operation descriptions. In all cases, however, it is shown how the UML semantic model can be conservatively extended to incorporate a more precise description.

Given the assumption that the UML semantics are incomplete and the debate about the suitability of meta-modelling as a formalization strategy, a number of formalization approaches have been tested by the group: The first approach to the problem has been to make the notations more precise and amenable to rigorous analysis by *integrating* them with a suitable formal specification notation. A number of integrated OO and formal notations have been proposed (e.g., see [BFLP97, JK96]). Most works focus on the generation of formal specifications from less formal OO models. This can reveal significant problems that are easily missed in less formal analyses of the models. Furthermore, the formal specifications produced can be rigorously analyzed, providing another opportunity for uncovering problems. However, a serious limitation of this approach is that it requires an in-depth knowledge of the formal notation and its proof system. This is often a significant barrier to industrial use. Often the formal notation is perceived by industrial practioners as being too mathematical and low level to be of practical use, and the gap between the graphical notation and formal notation as being too large.

The second approach to the problem has been to extend formal notations with OO features, thus making them more compatible with OO notations. Several extensions exist in literature (e.g., Z++ [LH94] and Object-Z [Duk91]). However, although a rich body of formal systems have resulted, they are still too different from current industrial methods to be suitable for general industrial application. In addition, there is also a lack of available analysis tools.

The third approach has been to directly express the semantics of UML in a formal language such as Z. This has the advantage of enabling a formally precise and unambiguous model of the semantics to be written (Z is well defined and formal). However, it is difficult to decide on the criteria to be used in constructing the model. If the UML meta-model semantics is exactly adhered to, then any mismatch between the formal language and the meta-model language is likely to result in a large, cumbersome, translation. On the other hand, if one uses the meta-model as a requirements document it may be possible to produce a smaller, more elegant description of the semantics. Unfortunately, there is also the risk that the standard semantics may not be captured fully or are misinterpreted. Whichever strategy is employed, it is important to remember that the semantics of UML are part of a standard, and that to develop an alternative model (unless it a provides a miraculous simplification or considerably improves usability, e.g. through better composition, abstraction and refinement concepts) is unlikely to be well received by industry. Thus, to ensure the best chance of the work having an influence on industry, the most sensible approach would be to stay as close to the standard as possible.

The final approach used by France, has concentrated on formalizing specific OO modelling concepts. This has the advantage of making the formalization process more manageable and also has helped to identify concepts which are particularly weakly defined in OO methods.

Taken together, the experience of using these different approaches has helped members gain a valuable understanding of the limitations and advantages of the present UML semantics. This will form the cornerstone for work being carried out to develop formal development methods, analysis and refinement techniques for UML.

3 Refinement

Once a formal definition is available it is possible to use it as foundation for developing formal refinement techniques for UML. Refinement has the same meaning in UML as for any formal language: it is the process by which an abstract model of a system (containing relatively little implementation detail) can be incrementally transformed into a model that can be readily implemented in a specific programming language. At each stage the correctness of the more detailed model must be verified against the abstract model. The most obvious way to achieve this is to check that each more detailed model satisfies the conditions necessary for it to be a valid refinement of its predecessor. Of course, this requires that the conditions for refinement have been precisely laid down.

Another approach is to pre-verify commonly used design steps. For example, a common step in the refinement of a many to many association is to resolve it with a many to one and one to many association. Given that the refinement conditions are known, a simple set of conditions can be given for ensuring that the step is valid with respect to the model semantics. Thus, such design steps may be used with a much reduced knowledge of formal proof techniques than the 'invent and verify' approach described previously.

Where UML does differ from other (formal) languages is in its use of diagrams. Whereas refinement in a textual formal language (like Z) involves manipulating textual syntax, refinement in UML is most naturally thought of in terms of *diagrammatical* manipulations. In other words, the refinement of a UML model implies a process of diagrammatical transformation. Moreover, because UML uses different diagrams to model a complete system, different transformations must be applied to different diagrams in order to effect a complete refinement process. Understanding the nature of these transformations and how they can be constructed in a compositional manner is currently an important issue being investigated by the group.

4 Deduction

As stated previously, UML provides a number of diagrammatical modelling tools for describing systems. Each diagram is intended to provide a different 'view' of the complete system model. For example, a class diagram is used to describe static (data) properties, whilst a sequence diagram aims to describe system behaviour. Although these diagrams are visually intuitive, their informality makes it difficult to rigorously deduce properties about the models they represent.

Just as with refinement, deduction can be thought of as a transformational process [Eva98a]. If a diagram, representing the desired properties of another diagram, can be obtained by applying a set of correctness preserving transformations then the property can be proven valid. In addition, it is often the case that properties expressed using the constraint language can be visualised as a diagrammatical object. Thus, it is possible to *visually* verify the correctness of the property with respect to the original diagram. Again, it is possible to develop a set of basic deductive transformations on UML diagrams which can be used as part of a rigorous analysis process for UML.

5 Conclusion

This short note has briefly described some of the issues being investigated by the precise UML group. As this work has progressed, it is increasingly been the case that members have sought to view UML as a formal language in its own right. Thus, an on-going goal has been to develop generic proof and refinement techniques which manipulate the basic components and abstractions of UML rather than those derived by language translation.

Appendix

This appendix briefly describes work that is being carried out individually and in cooperation by members of the precise UML group.

Imperial College, UK

At Imperial College current work is focussed on the semantics of dynamic modelling notations in the UML, such as interactions and sequence diagrams, using Real-time Action Logic (RAL).

We are also working on the development of a method for using the UML in conjunction with formal transformations. For highly critical applications (systems where the consequence of incorrect functioning may include loss of life or severe financial loss), it is important that the development process used can help detect and eliminate errors. The process should in particular support the *verification* of refined models against abstract models by comparing their semantics.

Our proposed process can be summarised as follows:

1. Requirements – modelled using Yourdon context diagrams and UML use case diagrams (without dependencies between use cases).

2. Essential Specification – described using UML class diagrams, operation schemas (from Fusion and Octopus) and statecharts.

3. Design – modelled using UML class diagrams, statecharts, sequence diagrams and collaboration diagrams.

The verifiable relationships between these models are:

1. Each input event/message on the system context diagram should have a system response described by an operation schema.

2. The effect described by the operation schema for an event e must be established by the completed response sequence to e described in design level statecharts, that is, by the transitions specified for e and the set of their generated events and transitions.

   A refinement relationship should be defined between the abstract state used in the operation schemas, and the state used in the statecharts.

3. Design level class diagrams should enforce all the properties asserted in the specification level class diagrams.

4. Sequence diagrams should be consistent with collaboration diagrams: the structure of object inter-calling should be the same.

5. Collaboration diagrams should be consistent with statecharts: messages sent by an object in response to a message m should correspond to events generated from transitions for m in the statechart of the object.

University of Munich, Germany

In Munich Bernhard Rumpe is elaborating the notion of refinement and composition in the context of UML diagrams and especially behavior descriptions. As textual notations and formalisms show, a notation becomes more useful, if it cannot not only be used to capture knowledge gained from analysis or decided during design, but if manipulation techniques exist that allow to deal with pieces of knowledge. Manipulating diagrams for example allows to

- iteratively add information and such provide more details of the system under consideration (refinement),
- transform one model into another one, which exhibits new properties or is more oriented towards an implementation (e.g. mapping State Diagrams into Code, mapping Sequence Diagrams into State Diagrams, or transforming between Sequence and Collaboration Diagrams) (view transformation)
- derive pieces of information from a given, detailed model, where the pieces of interest have been hidden by other details (derivation).

This work e.g. includes a refinement calculus for State Diagrams in several timed and untimed variants [Rum96, PR97a, RK96, PR94], refinement techniques for Sequence Diagrams [BGH+97a], and for Class Diagrams [BHH+97, Rum96]. Beyond the UML, some work was done manipulate and especially refine dataflow networks [PR97b]. The work is based on a rigourous, precise definition of what the kinds of target systems are [RKB95, KRB96].

Based on the experiences gained from these works, some general conclusions about the usefulness of and the path to a formal semantics of UML [Rum98, EFLR98, BGH+97b, BGH+98, BHH+97] have been drawn. One important conclusion is, that people will not use formality in a sense that diagrams (regarded as informal) are transformed into a set of logic formulae (regarded as formal). If the principles of refinement, composition, proof derivation, etc. developed in various logics shall be used in the context of UML, then these techniques have to be transfered into techniques for the UML diagrams. E.g. a rule allowing to split a state in a State Diagram in such a way, that the overall described behavior will persist, is a valid “semantic” refinement rule.

Formalization, e.g. as a mapping from a diagrammatic notation into a well known semantic domain is a first important step into this direction, but not useful in itself.

Universities of Boca Raton, and Colorado, US

At the University of Boca Raton, Florida, previous research led by Robert France focused on integrating UML with some of the more mature formal specification notations, for example, Z and Object-Z (e.g., see [SF97, SF97]). The process model supporting the use of integrated techniques that was developed required that developers first develop the OO models and then transform them to Z or Object-Z specifications. The generated formal specifications could then be used to support rigorous analysis of the models (via theorem-proving or animation), and they can provide a more precise specification of desired behavior.

The integrated techniques developed have been applied by students in a graduate program and on a case study [FB98] with some success, but their application also raised some issues that could deter their industrial uptake. A major concern is that such techniques require OO modelers to be familiar with both OO notations and formal specification notations if they are to be used effectively. Developers that are not familiar with formal notations can find it difficult to relate the mathematical concepts on which formal specifications are built to elements in their OO models. For this reason, carrying out analyses and interpreting analysis results can be problematic. One solution to this problem is to have modeling teams that consist of both experts in the OO and formal notations, but this may not be cost-effective for some organizations.

Discussions that led to the formation of pUML concluded that a formalization of UML should not require that developers manipulate mathematical expressions that are extraneous to their OO models. Rather, an attempt should be made to evolve UML to a formal language by associating a precise semantics with the notation that can be used as the basis for reasoning about the models. Such reasoning should require the developer to manipulate only the OO models they created. As a first step we are developing precise characterizations of what we consider basic OO modeling concepts as described in the popular literature and embedded in OO CASE tools. To date we have developed characterizations of the notion of aggregation [SFLP98] and have suggested extensions to the UML that can be used to more precisely capture the form of aggregation required [SLPFE98]. The characterizations should give us a good foundation on which a formal semantics for UML concepts can be built. This work is being carried out at Colorado University.

Pau University, France

We are developping an environment for the specification of complex software systems that is based on the Formal Specification Technique (FST) Z. This environment supports the development of precise and analyzable structured specifications of the desired behavior, and the validation of qualitative and quantitative properties. Specifications are developed using an integrated Z and object-oriented method (Fusion). The object-oriented method provides the structural constructs needed to manage the complexity of the model building activity.

In our approach, both structural and dynamic aspects are modeled using a unique formal notation. This environment supports the analysis of graphical, structured, and precise models of complex systems. This work was partially funded by NSF grant CCR-9410396.

York University

At York University, work is being undertaken by Andy Evans to develop a sound formal semantics and deductive system for UML diagrams based on the standard UML semantics. It is hoped to develop a diagrammatical deductive system for UML which can be used to verify important properties of UML models. In addition, work is beginning on the development of a rigorous, real-time development method for UML, supported by the real-time systems group at York.

References

[BFLP97] J.M. Bruel, R.B. France, and M. Larrondo-Petrir. Case-based rigorous object-oriented methods. In A.S.Evans and D.J.Duke, editors, *Proceedings of the 1st Northern Formal Methods Workshop*, Springer eWiC series, 1997.

[BGH+97a] Ruth Breu, Radu Grosu, Christoph Hofmann, Franz Huber, Ingolf Krüger, Bernhard Rumpe, Monika Schmidt, and Wolfgang Schwerin. Exemplary and complete object interaction descriptions. In Haim Kilov, Bernhard Rumpe, and Ian Simmonds, editors, *Proceedings OOPSLA'97 Workshop on Object-oriented Behavioral Semantics.* TUM-I9737, 1997.

[BGH+97b] Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Towards a precise semantics for object-oriented modeling techniques. In Jan Bosch and Stuart Mitchell, editors, *Object-Oriented Technology, ECOOP'97 Workshop Reader.* Springer Verlag :NCS 1357, 1997.

[BGH+98] Ruth Breu, Radu Grosu, Franz Huber, Bernhard Rumpe, and Wolfgang Schwerin. Systems, views and models of UML. In Martin Schader and Axel Korthaus, editors, *The Unified Modeling Language, Technical Aspects and Applications*, pages 93–109. Physica Verlag, Heidelberg, 1998.

[BHH+97] Ruth Breu, Ursula Hinkel, Christoph Hofmann, Cornel Klein, Barbara Paech, Bernhard Rumpe, and Veronika Thurner. Towards a formalization of the unified modeling language. In *Proceedings of ECOOP'97.* Springer Verlag, LNCS, 1997.

[BJR98] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language - a reference manual.* Addison Wesley, 1998.

[Boo94] G. Booch. *Object Oriented Analysis and Design with Applications.* Benjamin/Cummings Publishing Company, Inc., 1994.

[Duk91] D. Duke. *Object-Oriented Formal Specification.* PhD thesis, 1991.

[EFLR98] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Developing the UML as a formal modelling notation. In Pierre-Alain Muller and Jean Bezivin, editors, *UML'98 Beyond the notation. International Workshop Mulhouse France.* Ecole Superieure Mulhouse, Universite de Haute-Alsace, 1998.

[Eva98a] Andy Evans. Reasoning with UML class diagrams. In *Proceedings of WIFT'98.* IEEE, 1998.

[Eva98b] Andy Evans. UML class diagrams - filling the semantic gap (draft). Technical report, York University, 1998.

[FB98] Robert B. France and Jean-Michel Bruel. Applying fusion/UML to the invoicing problem. In *Proceedings of the Int. Workshop on Comparing Systems Specification Techniques, Nantes, France, eds. Michel Allemand, Chritian Attiogbe and Henri Habrias, Institut de Recherche en Informatique de Nantes*, 1998.

[Jac93] I. Jacobson. *Object-Oriented Software Engineering - a Use Case Driven Approach.* Addison Wesley, 1993.

[JK96] D. Randolph Johnson and H. Kilov. Can a flat notation be used to specify an oo system: using Z to describe RM-ODP constructs. In Elie Najm and Jen-Bernard Stephani, editors, *Proceedings of the 1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems.* Chapman and Hall, 1996.

[KRB96] Cornel Klein, Bernhard Rumpe, and Manfred Broy. A stream-based mathematical model for distributed information processing systems - syslab system model -. In Elie Naijm and Jean-Bernard Stefani, editors, *FMOODS'96, Formal Methods for Open Object-based Distributed Systems.* ENST France Telecom, 1996.

[LH94] K. Lano and H. Haughton. The z++ manual. Technical report, Imperial College, 1994.

[PR94] Barbara Paech and Bernhard Rumpe. A new concept of refinement used for behaviour modelling with automata. In *FME'94, Formal Methods Europe, Symposium '94.* Springer, 1994.

[PR97a] Barbara Paech and Bernhard Rumpe. State based service description. In John Derrick, editor, *Formal Methods for Open Object-based Distributed Systems.* Chapman-Hall, 1997.

[PR97b] Jan Philipps and Bernhard Rumpe. Refinement of information flow architectures. In M. Hinchey, editor, *ICFEM'97 Proceedings, Hiroshima. Japan.* IEEE CS Press, 1997.

[RBP$^+$91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[RK96] B. Rumpe and C. Klein. Automata describing object behavior. In H. Kilov and W. Harvey, editors, *Specification of Behavioral Semantics in Object-Oriented Information Modeling, P. 265-286.* Kluwer Academic Publishers, 1996.

[RKB95] B. Rumpe, C. Klein, and M. Broy. Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - Syslab Systemmodell -. Technical Report TUM-I9510, Technische Universität München, March 1995.

[Rum96] Bernhard Rumpe. *Formale Methodik des Entwurfs verteilter objektorientierter Systeme.* Tum doktorarbeit, Technische Universität München, 1996.

[Rum98] Bernhard Rumpe. A note on semantics (with an emphasis on UML). In Haim Kilov and Bernhard Rumpe, editors, *Second ECOOP Workshop on Precise Behavioral Semantics.* Technische Universit"at M"unchen TUM-I9813, 1998.

[SF97] Malcolm Shroff and Robert B. France. Towards a Formalization of UML Class Structures in Z. In *Proceedings of COMPSAC'97*, August 1997.

[SFLP98] Monika Saksena, Robert B. France, and Maria M. Larrondo-Petrie. A characterization of aggregation. In *Proceedings of the 5th International Conference on Object-Oriented Information Systems (OOIS'98)*, 1998.

[SLPFE98] Monika Saksena, Maria Larrondo-Petrie, Robert France, and Matthew Evett. Extending the notion of aggregation in UML. In *Proceedings of the UML'98 International Workshop*, 1998.