Verifying OCL specifications of UML models : tool support and compositionality

Kyas, M.

Chapter 8

Conclusion

In the introduction we have written that we develop a formal semantics for a subset of UML in this dissertation, which allows the formal validation of real-time embedded systems modelled with UML and specified with OCL. To achieve this, we have developed a formal semantics for UML class diagrams, object diagrams, and OCL, suitable for an embedding into the theorem prover PVS. The embedding uses the formal semantics of state machines developed by van der Zwaag and Hooman [149].

8.1 Summary

This dissertation represents a further step towards a rigorous formalisation of UML and towards OCL for tool supported formal verification of models. Our results extend, update, or complement the results obtained by Mark Richter [133] and Demissie Aredo [6] in various directions:

- We have described a formal semantics of UML class diagrams and UML object diagrams. This semantics establishes object diagrams as models of class diagrams and constraints as invariants on object diagrams. Multiplicities of associations are understood as constraints, that is, as invariants of object diagrams.
- In currently available UML case tools even basic support for OCL is missing. Some tool vendors provide plug-ins which validate OCL constraints, but they are mostly used during detailed design; constraints are used in the same way assertions are used in programming languages. One reason for this we have identified is the inflexibility of the type system of OCL. To alleviate this, we have proposed a generalisation of this type system, demonstrated that it offers greater flexibility in developing the underlying model, and implemented a type checker based on this extension.
- OCL is a constantly evolving language. We have provided a formal semantics for OCL 2.0, which we have formalised in the language of the theorem prover PVS. This formalisation is supported by a translator, which translates UML models and OCL constraints into PVS. The translator implements a shallow embedding

of OCL into PVS. The feasibility of this approach has been demonstrated by verifying case studies, among others, the Sieve of Eratosthenes, using this tool.

- We have analysed the newly introduced *OCL Message Expressions*, and found them lacking, because they only allow to specify whether a message has been sent during the execution of an operation call. We have proposed to include histories into OCL, which allow to specify whether messages have been sent *or received* and to reason about the order of these. We have demonstrated that all OCL message expressions can be expressed in terms of OCL constraints on histories. Therefore, histories form a conservative extension of OCL. Histories, however, offer much greater flexibility, in that these histories may also be used in class invariants and preconditions of operations.

- We analysed the implications of object creation for a history-based formalism. This has led to the axiomatisation of histories with object creation, which we have formalised in the PVS theorem prover. We show that in the case of reactive systems, that is, systems in which each action is the reaction to a message received, it is not necessary to observe object creation. A safe approximation of the tree of creation can be computed from histories. If objects are active, we observe *prenatal object activity,* that is, objects initially appear to be root objects (which have not been created by any object) but later one may observe that they have indeed been created by other objects. Using this axiomatisation we have proved a case study, the Sieve of Eratosthenes, correct.

- The final extension of the trace based formalism was to extend our notion of communication traces with time. We have applied this method to the error logic component of the medium altitude reconnaissance system (MARS) in order to derive a correct specification of this component.

8.2 Future Research

Formalising the UML is very challenging, not only because of the sheer size of the standard, its number of concepts and notations. Even the designers of UML have expressed the opinion that there is too much overlap and redundancy in the notations of UML. The reason is that the Object Management Group not only formalises best practises, but also suggests new notations and new technology, which become part of the standards, without having proved their usefulness in practise. It has been suggested by Cris Kobryn, who chaired the standardisation of UML 2.0, that UML 3.0 should only contain the parts of UML 2.0 which have proved their usefulness in practise [80]. Therefore, formalising every aspect of the UML should *not* be a goal for future research.

Instead, one should focus on identifying a *small* selection of notations from UML and develop a formally sound system development method around these notations. The notations we selected for this dissertation are quite powerful for modelling event-driven systems. Architectural diagrams may be added, because they display aspects of class diagrams and object diagrams in an integrated manner.

In this dissertation we have not considered the formalisation of state machines. Instead, we used the semantics proposed by van der Zwaag and Hooman [149]. This semantics differs from the semantics described in the UML standard [111]. A semantics has been derived from the standard by Schönborn [140]. This semantics may be formalised in PVS and used with our tool. This would result in an integrated tool that allows the formal verification of UML 2.0 models in PVS.

The methods described in this dissertations may be extended with refinement-based methods. This extension supports a top-down development of systems driven by formal development steps.

In order to make the UML and the OCL useful, strong semantical analysis tools and validation tools have to be developed. Without these tools, the UML and especially OCL is only a write-only specification formalism, and consequently a weak formal method in the sense of Wolper [156]. A specification is called write-only, if it is only written for the sake of compliance with standardised development methods, but never used for the development of the specified system.

In this dissertation we have analysed the light-weight end of these methods, namely type-checking, and the heavy-weight end, namely interactive theorem proving, without covering the methods in between these extremes. Especially strong static analysis methods have the potential of improving the design of safety critical systems by detecting errors early.

Because the risks involved in designing and implementing embedded real-time systems, which are very often safety critical, strong formal methods are needed, which allow the rigorous analysis and formal verification of models, from which eventually a working system has to be designed. This is one of the directions suggested by Grady Booch [12], who is one of the fathers of UML:

> Note that the OMG speaks of MDA (Model-Driven Architecture). In my world view, the UML can be much more than just a means of constructing systems: for example, consider debugging a system at the level of UML diagrams.

This dissertation provides an approach to verifying UML diagrams, and through this also a way for debugging UML diagrams. But one draw-back of our approach is that it needs a lot of expertise on using the theorem prover *and* the translator in order to trace a detected error back to its location in the original diagram. Therefore, one direction of research is to achieve seamless integration between theorem proving and the original diagram, perhaps by defining a formal deductive system for OCL and build a proof

checker for this deductive system. The major obstacle to overcome is to find deductive rules for late-binding of functions.