Lessons Learned From Microkernel Verification

Bernhard Beckert[1] **and Thorsten Bormer**[2*]

[1] beckert@kit.edu, http://formal.iti.kit.edu/~beckert/
[2] bormer@kit.edu, http://formal.iti.kit.edu/~bormer/
Institute for Theoretical Informatics
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract: Software verification tools have become a lot more powerful in recent years. Even verification of large, complex systems seems feasible, as demonstrated in the L4.verified and Verisoft XT projects. Still, functional verification of large software systems is rare. In this paper we hint at some issues that may impede widespread introduction of formal verification in the software lifecycle process.

Keywords: software specification, software verification

Here we report on lessons learned from microkernel verification: within the Verisoft XT project, core parts of the embedded hypervisor PikeOS (see http://www.pikeos.com) have been verified [BBBB09] using the VCC tool (see http://vcc.codeplex.com). While PikeOS is several orders of magnitude smaller than, e.g., the Linux kernel, for verification purposes this is a substantial code size.

Specification and verification of large software systems does not scale linearly with the number of methods, i.a., due to the interactions between methods operating on parts of a shared program state. There are several ways to simplify verification: (a) reduce the cost of specifying and verifying a single property of a single function, (b) decompose the verification task by verifying one module of the system at a time and (c) abstracting from details of the system's implementation.

All three of these points have already been addressed to a certain extent by current deductive verification tools. Regarding goal (a), verification tools have made a leap forward in recent years, enabling users to verify individual functions once considered challenging with ease. Towards feature (b), annotation-based verification tools like VCC already make use of decomposition of the verification task by verifying methods, as well as threads, in a modular fashion. Concerning the last point (c), abstraction is possible using a separate specification state and support for abstract data types. Still, support for modularization and abstraction has to be better supported by the verification systems in order to be helpful for the verification of large software systems.

In the following, we will illustrate why verification of a system like PikeOS is still challenging, despite all support by the verification tool and methodology. Although some characteristics of PikeOS are prominent for microkernels, they are in no way exclusive for this type of software.

Modularization. Modularization reduces verification effort by decomposing the verification task, but it does only help to a limited amount in finding the right auxiliary specifications needed for verification. In addition, architectures such as microkernels feature some inherent characteristics that restrict the extent to which the specification task can be modularized.

* Work partially funded by the German Federal Ministry of Education and Research (BMBF) in the framework of the Verisoft XT project under grant 01 IS 07 008. The responsibility for this article lies with the authors.

In the case of PikeOS, single C functions are deliberately kept simple to facilitate maintainability and certification – the functionality of the kernel is rather implemented by interaction of many of these functions, operating on shared data. Microkernels, as all operating systems, have to keep track of the overall system's state, resulting in large and complex data structures. As a result, method specifications have strong dependencies on each other, with several consequences:

(1) Finding the right annotations for a single method requires the verification engineer to consider several methods at once. Feedback given by annotation-based verification tools in case of a failed verification attempt so far only focuses on the method currently being verified. Further measures are needed to help the user in case of analyzing problems with dependent specifications.

(2) Dependencies between methods obfuscate module boundaries and thus makes finding the right module interfaces difficult. In addition, even if larger modules can be identified, there is no particular support for a hierarchical modularization in annotation based verification systems in order to specify properties of such a module, e.g., on the system architecture level. Also for concurrent systems, a fixed modularization granularity may not be appropriate: parts of the implementation to be executed in an atomic fashion may span across several methods.

Abstraction. To find the right abstraction for data structures in the system the source code alone is often not sufficient in practice. Gathering the important properties of the data structure is crucial in order to find the right abstraction. While some techniques such as CEGAR exist that may help in some cases in finding the right abstractions, these methods are not sufficiently supported in deductive annotation-based systems. Again, dependencies between methods that operate on shared data complicate finding the right abstraction – apart from support by verification tools information from system developers and architects is vital in this case.

Conclusion. For efficient verification of large software systems, we claim that better support from verification tools is needed to scale verification from individual modules to whole software systems. This would have to include support in finding the right modularization and abstraction. A first step is to provide the user with feedback in case of interdependent method specifications, so that mismatching contracts are discovered early in the specification process [BBMS11].

Besides annotation-based specifications that are well suited for description of functions on code level, there is need for further specification constructs tailored for description of the system properties on higher levels of abstraction. These formalisms should also allow to integrate knowledge of system developers. On the code level, we propose to use existing formalisms for abstract data types like CASL, to be able to concisely specify commonly used data structures [BBK12].

Bibliography

[BBBB09] C. Baumann, B. Beckert, H. Blasum, T. Bormer. Formal Verification of a Microkernel Used in Dependable Software Systems. In Buth et al. (eds.), *SAFECOMP'09*. LNCS 5775, pp. 187–200. Springer, 2009.

[BBK12] B. Beckert, T. Bormer, V. Klebanov. Improving the Usability of Specification Languages and Methods for Annotation-Based Verification. In Aichernig et al. (eds.), *Formal Methods for Components and Objects*. LNCS 6957, pp. 61–79. 2012.

[BBMS11] B. Beckert, T. Bormer, F. Merz, C. Sinz. Integration of Bounded Model Checking and Deductive Verification. In *FoVeOOS'11*. Pp. 86–104. 2011.