Introspection-Based Verification and Validation

Hans P. Zima
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109
E-mail: *zima@jpl.nasa.gov*

Abstract

This paper describes an introspection-based approach to fault tolerance that provides support for runtime monitoring and analysis of program execution. Whereas introspection is mainly motivated by the need to deal with transient faults in embedded systems operating in hazardous environments, we show in this paper that it can also be seen as an extension of traditional V&V methods for dealing with program design faults.

Introspection—a technology supporting runtime monitoring and analysis—is motivated primarily by dealing with faults caused by hardware malfunction or environmental influences such as radiation and thermal effects [4]. This paper argues that introspection can also be used to handle certain classes of faults caused by program design errors, complementing the classical approaches for dealing with design errors summarized under the term *Verification and Validation (V&V)*.

Consider a program, P, over a given input domain, and assume that the intended behavior of P is defined by a formal specification.

Verification of P implies a static proof—performed *before execution* of the program—that for all legal inputs, the result of applying P to a legal input conforms to the specification. Thus, verification is a methodology that seeks to *avoid* faults. *Model checking* [5, 3] refers to a special verification technology that uses exploration of the full state space based on a simplified program model.

Verification techniques have been highly successful when judiciously applied under the right conditions in well-defined, limited contexts. However, in general they face a number of theoretical and practical challenges and limitations including the following:

- Theoretical Limits: Undecidability and NP-completeness
- Scalability Challenges: State Space Explosion
- Requirement Specification Challenge: Incompleteness

A second major V&V technique is **test**, executing a program for a specific set of inputs. A recently developed test methodology executes test cases and checks the properties of the executing program using monitoring of events and data values based upon an instrumentation of the target program [2]. However, as recognized by Edsger Dijkstra as soon as 1972, tests can prove the *existence* of an error but never the *absence of all* errors [1].

In summary, V&V technology cannot, for theoretical as well as practical reasons, provide a complete solution to the problem of proving the correctness of programs. As a consequence, it is important to realize the fact that *design errors do occur* and to develop methods for dynamic recovery if they actually happen. This is where introspection comes into play. Furthermore, it is important to note that introspection technology is able to detect *anomalies* in an execution that are not explicit faults, such as unusual execution times for a loop or borderline values of variables. Such occurrences may not trigger a direct action but they may be stored in a knowledge base for later retrieval.

The full paper will discuss the introspection framework developed in a research project conducted at JPL [4] and describe methods for using static analysis, profiling, and dynamic analysis to create assertions that perform runtime verification [6]. In addition, we will outline an approach that integrates introspection with the program design methodology.

References

[1] Edsger W. Dijkstra. Notes on Structured Programming. In O.-J. Dahl, Edsger W. Dijkstra, and C.A.R. Hoare, editors, *Structured Programming*, pages 1–82. Academic Press, London, UK, 1972.

[2] Klaus Havelund and Allen Goldberg. Verify Your Runs. In *Proceedings Verified Software: Theories, Tools, Experiments (VSTTE'05)*, October 2005.

[3] Gerard J. Holzmann. *The SPIN Model Checker. Primer and Reference Manual*. Addison-Wesley, 2003.

[4] Mark L. James and Hans P. Zima. An Introspection Framework for Fault Tolerance in Support of Autonomous Space Systems. In *Proceedings 2008 IEEE Aerospace Conference*, March 2008.

[5] Masoud Mansouri-Samani, Corina S. Pasareanu, John J. Penix, Peter C. Mehlitz, Owen O'Malley, Willem C. Visser, Guilleaume P. Brat, Lawrence Z. Markosian, and Thomas T. Pressburger. Program Model Checking. A Practitioner's Guide. Technical report, Intelligent Systems Division, NASA Ames Research Center, April 2007. Version 1.0.

[6] Hans P. Zima and Barbara M. Chapman. *Supercompilers for Parallel and Vector Computers*. ACM Press Frontier Series, 1991.