Software Fault Protection for Avionics

Allen Goldberg
Kestrel Technology
goldberg@kestreltechnology.com

Greg Horvath
JPL
Gregory.A.Horvath@jpl.nasa.gov

In this position paper we present an approach for achieving high assurance of flight systems that is related to and complementary with software verification and validation methods. We propose a model-based approach to runtime monitoring and error recovery of flight software, in short, *software fault protection*. More specifically certain aspects of the intended system behavior are formalized in a model. At runtime the software is monitored and compared with the model. Should there be disagreement, the model specifies a recovery action to perform. Like all V&V processes it is concerned with checking the consistency of the behavior of the software with some other entity such as a formal or informal model, requirements, or a set of test cases. In this case the consistency check is performed in operation rather than during development.

The value proposition for this approach is based on the following two observations:

1. It is generally acknowledged that almost all fielded software contains errors that are not uncovered, even after undergoing a rigorous verification and validation campaign. Often these errors are manifested by software behavior that is *obviously* incorrect. The software may crash, fail to meet its interface requirements, consume an unexpected amount of system resources, or produce a result that is clearly erroneous given the constraints of the application. Due to the obvious nature of these types of failures, a relatively simple model of software behavior may be developed in order to detect, at run time, errors that may only manifest themselves in rare situations. As a result of the rarity of such behaviors, these situations are often unanticipated and, therefore, not tested. Recovery from such errors may be possible because the transient conditions that caused them are no longer in force.
2. It is our view that a model that effectively detects such failures describes interface behavior among components, computational resource usage and data reasonableness. These attributes are somewhat crosscutting to the explicit functional behavior of a component, thus minimizing common errors in the model and code. Such a model is inexpensive to monitor, easy to validate/certify, mitigates adoption risks, and enables an attractive technology insertion path. .By concentrating on developing a strategy that attempts to simply detect that an error has occurred rather than attempting to pinpoint the exact cause of the error, we believe that software fault protection will be a suitable complement to V&V processes, contributing to more reliable software.

Our approach is for integrated hardware/software fault protection that builds upon and extends ARINC 653. The ARINC 653 standard defines an Application Executive, or

ApEx, which provides OS and middleware services for Integrated Modular Avionics. Central to the facilities provided by an ARINC 653 compliant ApEx are temporal and spatial partitioning. This partitioning insures that faults do not propagate beyond the partition and thus simplifies the task of fault isolation. Additionally, the ARINC 653 standard defines a health monitoring capability, intended to detect and respond to both hardware and software faults at the process, partition, module, and system level. This facility is customizable with user code that runs in its own "health management" partition or as a high-priority process within an application partition.

An ARINC 653 is configured by XML file that provides description of the partitions, their resources (memory, and the schedule for processor usage), identifies the processes executing in each partition, and the ports and channels used for the two forms (queuing and sampling) of message-based communication defined at the partition level. It tabulates fault response to errors detected by the hardware, the system software or application code. When a fault occurs a user-supplied fault handling procedure can be invoked that can log errors, restart processors or partitions, stop and/or replace processes, or report the error to a higher level entity (e.g. from partition to module).

We note that the configuration file is a model of architecture (components and their connections) resource usage, and health monitoring response, and as such is a primitive version of our fault protection o model. Thus a natural and incremental adoption strategy is to extend this model with a finer description of intended system behavior and then generate (at least some of) the system monitoring, model execution, and recovery code from the model. We believe only the most conservative recovery strategies should be employed. These include logging errors, restarting components at the process, partition, and module level, entering a "safe mode" or replacing a component with one that performs only essential services.

In summary ARINC 653 is an excellent starting point for a software fault protection system. We are currently designing such a capability.