Goal-Based Flight Software Health Management Services

Matthew Barry
Kestrel Technology, LLC
3260 Hillview Avenue
Palo Alto, CA 94304
Email: mrbarry@kestreltechnology.com

Gregory Horvath
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
Email: gregory.a.horvath@jpl.nasa.gov

EXTENDED ABSTRACT (REVISED)

The NASA-sponsored FAILSAFE project is developing concepts and prototype implementations for software health management in mission-critical real-time embedded systems. The project unites features of the industry standard ARINC-653 Avionics Application Software Standard Interface and the Jet Propulsion Laboratory's Mission Data System (MDS) technology. The ARINC-653 standard establishes requirements for the services provided by partitioned real-time operating systems. The MDS technology provides a state analysis method, canonical architecture, and software framework that facilitates the model-based design and implementation of software-intensive systems. Our conjecture is that a model-based development and run-time system that is both state-centric and goal-directed offers compelling features and capabilities for real-time safety-critical health monitoring.

In a related prototype implementation, we are using the MDS technology to provide the health management function for an ARINC-653 application implementation. In particular, we are showing how this combination enables reasoning about and recovering from application software problems. In order to make it a compelling demonstration for current aerospace initiatives, we imposed on our prototype a number of requirements derived from NASA's Constellation Program. In particular, we adopted both the computer-based control system safety requirements and the safety-related requirements completeness checks from the Constellation Program's computing system requirements. The control system safety requirements address issues associated with loss of function and with inadvertent activation. The completeness checks address a number of issues associated with the specification and implementation of features for software safety and for the interaction of hardware and software. For each relevant element in both sets of adopted requirements, we identified one or more demonstration features that might show how the requirement might be met using a software health management approach. Our prototype application software mimics the Space Shuttle orbiter's abort control sequencer software task, which provides safety-related functions to manage vehicle performance during launch aborts. We turned this task into a goal-based function that, when working in concert with the software health manager, aims to work around software and hardware problems in order to maximize abort performance results.

In addition to our adopted application requirements, the ARINC-653 standard imposes a number of requirements on the system integrator for developing the requisite error handler process. Under ARINC-653, the health monitoring (HM) service is invoked by an application calling the application error service or by the operating system or hardware detecting a fault. The recovery action depends on the error level. The system integration specifies in the module HM and partition HM tables the recovery actions for each module and partition level error. The application programmer defines in a special error handler process the recovery actions for process-level errors. The error handler can stop and restart failed processes, restart the entire partition, or shut down a partition. It is these HM and error process details that we implement with the MDS technology, showing how a state-centric approach is appropriate for identifying fault determination details, and showing how the framework supports acting upon state estimation and control features in order to achieve safety-related goals.

NASA's recent Constellation Program computing system requirements emphasize software safety considerations across the product life cycle. Many of these considerations have to do with the identification and avoidance of hazardous states, and of the control of the system when entering or leaving a hazardous state. There are many requirements having to do with loss of function related to hazardous controls, and with the inadvertent activation of functions that may introduce hazards. Implementations of the controls for these requirements typically involve locks, inhibits, confirming actions, independent control paths, and so on. Each of these considerations is difficult to deal with individually, at the function level, at any point in the product life cycle. Nevertheless, they share the notion of reasoning about state. The overall intention of the requirements and the system that realizes the requirements is to maintain a goal of preserving safe state. We believe that the MDS technology, which provides a state-based view of the system across the life cycle along with a run-time software framework that reasons about achieving goals with respect to these states, offers an elegant engineering solution that addresses these software safety concerns. Implemented as a health monitor, its goal would be to preserve safety by monitoring and controlling the state of safety-related controls. If given more authority, as in our FAILSAFE prototype, it would also monitor and control application software so as to achieve application performance goals. Tactics chosen at run-time to accomplish goals would be designed to fulfill the many procedural aspects of the control system safety requirements. Ideally, for complex systems both the system under control and its health monitor would be implemented in a model-based, state-based, and goal-based fashion.

Our workshop contribution would take up a few ideas for further investigation. First, the Constellation Program's computing system requirements endorse Leveson's work in ensuring the completeness of system safety requirements. Many of these requirements have to do with ensuring the completeness of understanding of the system state trajectory, such as avoiding or reducing the time spent in hazardous states. We believe there are certain meta-model constructs or modeling and programming idioms that would facilitate such analysis, and we know we can use model-checking technology to explore hypotheses on the model. We would like to develop the meta-model constructs, say by including rigorous design patterns that implement safety features (e.g. checkpoints, roll-backs, containment barriers, load shedding, and many others). These constructs may be realized within a *safety façade* for MDS, for example. Merely modeling the completeness requirements would go a long way toward introducing the system safety considerations in model-based development. Second, the model-based development techniques offer a convenient way to identify the data needed to enable run-time monitoring for enforcing properties of interest. The Constellation Program computing system requirements, for instance, identify a number of control system concerns expressible as safety properties. These properties, when modeled and implemented, should be monitored by the run-time health manager as a safety feature. Third, when working together across a partitioned run-time system, the application software and its health monitor must have cooperative performance in order to achieve the system goals. We would like to explore the architecture-level properties and quality attributes that enable the design-time analysis and run-time achievement of co-operative system goals. Contributing to this end, the MDS technology provides a way to compute during the analysis phase the performance characteristics of its run-time load given the number of MDS architectural elements (e.g. state variables, estimators, controllers, and hardware adapters), rate groups, and component operations in the design. The rigorous formal structure imposed by MDS yields simpler performance quantification and modeling which in turn yields complete and explicit measurement of CPU time and memory demand for the run-time deployment. Fourth, we believe it may be useful (and have started to develop) a reactive system specification of the ARINC 653 application executive system calls. Such a specification enables analysis of safety and liveness properties on the executive as well as applications using that executive. We would like to incorporate this specification into the executable model-based application development tool suites so that safety properties may be examined throughout the life cycle.

ACKNOWLEDGMENT

The FAILSAFE project is sponsored by a NASA Small Business Technology Transfer (STTR) Phase II award under contract NNA07AA24C.

A related paper by the same author describing a prototype implementation of these ideas was accepted for the SMC-IT 2009 conference.