Towards Automotive Software Health Management (SHM)*

Lee Pike† Galois, Inc.‡

February 15, 2011

Abstract

Integrated Vehicle Health Management (IVHM) covers frameworks for detecting, diagnosing, and mitigating faults in hardware and structures in aerospace systems. Software Health Management (SHM) applies the goals of IVHM to software-intensive systems to detect software faults in real-time and to mitigate them. In this position paper, we describe the needs and challenges of SHM in automotive systems.

1 From IVHM to SHM

Integrated Vehicle Health Management (IVHM) in aviation and space systems broadly covers tools and techniques for the detection, diagnosis, and mitigation of faults and failures during flight. Traditionally, IVHM has focused on hardware, including all aspects from structural integrity to mechanical systems to built-in electronics test. IVHM systems monitor the physical system and compare it against the mathematical model of the physical system. As noted by a National Research Council report, doing so allows the "aircraft to trace back the system anomalies through a multitude of discrete state and mode changes to isolate aberrant behavior" [4]. Such systems can lower maintenance costs as well as improve safety. An example is the Prognostics Health Management System designed for the Joint Strike Fighter.[1]

NASA has realized that traditional IVHM approaches fail to account for software failures. Software is an ever-growing source of guidance, navigation, and control systems in aircraft, as fly-by-wire becomes ubiquitous. Consequently, NASA has embarked on a *Software Health Management* (SHM) initiative to bring some of the advances of IVHM to software-intensive systems.[2]

IVHM typically diagnoses degrading performance of materials and hardware. But software does not degrade over time in the same way, so it should be reliable on its first use, and it should remain reliable. Evidence for the reliability of critical software comes principally in one of three forms: certification, testing, and formal verification.

*The research reported herein was supported in part by the NASA Aviation Safety Program under Contract #NNL08AD13T.

†Email: `leepike@galois.com`; personal webpage: `http://www.cs.indiana.edu/~lepike`; phone: +1 503 626 6616 ext. 135.

‡Address: 421 SW 6th Ave., Ste. 300, Portland, OR 97204, USA; `www.galois.com`.

[1] `http://ftp.rta.nato.int/public//PubFullText/RTO/MP/RTO-MP-079(II)///MP-079(II)-(SM)-41.pdf`

[2] `http://www.aeronautics.nasa.gov/nra_pdf/ivhm_tech_plan_c1.pdf`

For civilian aircraft, the Federal Aviation Authority (FAA) codifies high-assurance software development processes in standards such as DO-178B [6]. Currently, these standards rely heavily on a combination of rigorously-documented development processes and software testing. While certification provides evidence that good practices are followed in software development, it does not guarantee correctness of the end artifact. More or better testing can catch "easy to medium" software errors, but not deeper errors. Testing has been shown to be infeasible on its own for demonstrating reliability, at least for ultra-critical real-time software [1]. Consequently, The National Academies advocate the use of mathematical proof to augment testing [2].

Although *formal verification*—the mathematical proof of software systems—greatly increases confidence in software correctness, the complete verification of complex implementations remains elusive—the (full) formal verification of approximately ten thousand lines of code represents the state-of-the-art [3]. Formal verification can nevertheless be applied to abstractions of a system, but there is the risk that the abstractions do not hold.

Because of the respective disadvantages of certification, testing, and formal methods, a fourth form of evidence is being explored: the idea of *runtime monitoring* as applied to software. Runtime monitoring encapsulates techniques in which a simpler process (usually) software monitors a more complex one. Ideally, software monitors are synthesized directly from specifications and the synthesis ensures that monitors are inserted at correct points in the observed program.

2 Runtime Monitoring in Critical Embedded Systems

Ideally, the SHM approaches that have been developed in the literature could be applied straightforwardly to real-time embedded systems. Unfortunately, these systems have constraints violated by many previous approaches. First, a SHM approach for embedded systems must have fixed upper-bounds on timing and memory usage. Even assuming such upper bounds, there are more severe constraints, which we summarize using the acronym "FaCTS":

- *F*unctionality: the SHM system cannot change the target's behavior (unless the target has violated a specification).
- *C*ertifiability: the SHM system must not make re-certification of the target onerous.
- *T*iming: the SHM system must not interfere with the target's timing.
- *S*WaP: The SHM system must not exhaust size, weight, and power (SWaP) tolerances.

3 Copilot: A Hard Real-Time Runtime Monitor

Galois, Inc., under contract to NASA, has developed a new monitoring language called *Copilot* that is designed to satisfy the *FaCTS* properties [5]. Copilot monitors compile into constant-time and constant-space (i.e., no dynamic memory allocation) C programs. The language follows a sampling-based monitoring strategy in which variables or functions of an observed program are periodically sampled, and properties about the observations are computed.[3]

Copilot is designed to easily compose with existing embedded C code, even if there is no underlying operating system. Copilot monitors are functions that do not affect the state of the programs they are composed with, unless some monitored condition is violated. Generated monitors are C functions with their own hard real-time schedules that can be scheduled as a task in the overall system design.

[3]Copilot is released under the open-source BSD3 license, and up-to-date source code can be found here: `http://leepike.github.com/Copilot/`.

4 Challenges in Automotive SHM

While recent R&D in SHM for avionics provides some guidance, the automotive industry presents particular challenges to software health management that must be addressed:

- *Mean Time Between Maintenance*: Automobiles may be in-service for decades, and years may pass between maintenance calls. The manufacturer has less control over maintenance than commercial aircraft manufacturers. SHM with high mean time between maintenance may require new approaches, e.g., self-healing software systems.
- *Heterogeneity*: The software and hardware in modern automobiles is especially diverse, ranging from small microcontrollers to general-purpose systems in on-board entertainment and telemetry systems. A SHM approach must span the spectrum of software architectures.
- *Security*: The software systems in automobiles contain proprietary software from a variety of manufactures; a SHM approach may have to integrate monitors for such software without requiring those manufacturers to release protected IP. Additionally, mechanisms to monitor and mitigate software faults potentially provide vectors for introducing viruses or rootkits.
- *Cost*: The consumer market is very reactive to costs; a SHM approach cannot require significant additional hardware or infrastructure.

Author's Biography

Since 2005, Lee Pike has been research engineer at Galois, Inc., an R&D company that delivers software solutions for safety- and security-critical systems. At Galois, Dr. Pike has led multiple research projects in the areas of security, formal verification, and virtualization funded by NASA and the DoD. From 2003 to 2005, Dr. Pike was a staff scientist in the Formal Methods Group of the NASA Langley Research Center. He researched the design and formal verification of fault-tolerant real-time bus architectures. In 2003, Dr. Pike was also a visiting researcher at the National Institute of Aeronautics.

Dr. Pike holds a Ph.D in Computer Science from Indiana University, Bloomington (2006) and has published in the areas of formal methods and embedded systems. He has won a Best Paper award at the IEEE *Formal Methods in Computer-Aided Design (FMCAD)* in 2007, and has been quoted in *Flight International* magazine. A full list of publications can be found on Dr. Pike's personal webpage, `http://www.cs.indiana.edu/~lepike`.

References

[1] R. W. Butler and G. B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19:3–12, 1993.

[2] D. Jackson, M. Thomas, and L. I. Millett, editors. *Software for Dependable Systems: Sufficient Evidence.* National Academies Press, 2007.

[3] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal verification of an OS kernel. *Communications of the ACM (CACM)*, 53(6):107–115, June 2010.

[4] National Research Council. Decadal survey of civil aeronautics. National Academies Press, 2006.

[5] L. Pike, A. Goodloe, R. Morisset, and S. Niller. Copilot: A hard real-time runtime monitor. In *Runtime Verification (RV)*, volume 6418, pages 345–359. Springer, 2010.

[6] RTCA Inc. Software considerations in airborne systems and equipment certification, 1992. RCTA/DO-178B.