

Towards Software Health Management with Bayesian Networks

Johann Schumann
SGT, Inc., NASA Ames
Johann.M.Schumann
@nasa.gov

Ole J. Mengshoel
Carnegie Mellon University
Ole.J.Mengshoel
@nasa.gov

Ashok N. Srivastava
NASA ARC
Ashok.N.Srivastava@nasa.gov

Adnan Darwiche UCLA
darwiche@cs.ucla.edu

ABSTRACT

More and more systems such as aircraft, machinery, and cars rely heavily on software, which performs safety-critical operations. Assuring software safety through traditional V&V has become a tremendous, if not impossible task, given the growing size and complexity of the software.

We propose that SWHM (SoftWare Health Management) has the potential to increase safety and reliability of high-assurance software systems. SWHM can build upon the advanced techniques from the area of system health management to *continuously* monitor the behavior of software *during operation*, quickly detect anomalies and perform automatic and reliable root-cause analysis. Such a system would not replace traditional V&V, but rather supplement it. The information provided by the SWHM system can be used for automatic mitigation mechanisms (e.g., recovery, dynamic reconfiguration) or presented to a human operator for further analysis. SWHM may also feature a key prognostic capability, which can improve the reliability and availability of the software system because it provides information about soon-to-occur failures or looming performance bottlenecks. In this paper, we discuss research challenges associated with developing an SWHM system, and discuss how Bayesian networks (BN), a key technology used in advanced diagnostics systems may be used for SWHM modeling.

Categories and Subject Descriptors

D.2.5 [Software Engineering]: Testing and Debugging, Diagnosis

General Terms

Reliability

Keywords

Health Management, Bayesian Network, Verification and Validation

Copyright 2010 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.
Copyright 2010 ACM 978-1-4503-0427-6/10/11 ...\$10.00.

1. INTRODUCTION

In modern aircraft and other complex machinery, important electrical, mechanical, and hydraulic components and systems are managed by ISHM (Integrated System Health Management) systems. These can detect, diagnose, predict, and mitigate adverse events during the operation of the system and can help increase the reliability, dependability, and the life of the system while potentially also reducing maintenance costs. With the help of such diagnostic and prognostic techniques, appropriate mitigation strategies can be selected. Example mitigation strategies include replacement, repair, or switch to a redundant system.

Software Health Management, a relatively novel development, will extend this concept to software systems. An SWHM system continuously monitors the behavior of the software and the interfacing hardware or sensor components and can detect precursors to faults or failures in the system. Using an abstract model of the software (e.g., a Bayesian statistical model), the SWHM can detect unexpected behavior, reason about its root cause (failure identification), and perhaps even trigger failure repair or mitigation actions. A similar mechanism could be used for prognostic purposes, trying to reliably predict possible software and/or system failures in the future.

In principle, an SWHM system could operate similar to a traditional ISHM system, but would focus its attention on software instead of hardware. However, there are substantial differences between hardware and software systems:

- Many software errors do not develop when the software is in operation but are introduced at some stage of the software development cycle, as there are typically no 'wear-and-tear' effects in software. Examples include requirements errors, design flaws, or coding errors, just to mention a few. If these are not detected and removed during testing, they remain (dormant) in the software system and can show up during operation.
- Failures in software often occur due to problematic interoperation with hardware. Hardware systems (and their sensors) might behave differently than expected, and thus could cause software failure. Such a different behavior could be introduced by accident during development (e.g., a change in hardware is not reflected in the software design), as a result of hardware failure (e.g., a broken sensor cable, or a disabled sensor), or gradual degradation (e.g., signal noise increases be-

yond the specified level and causes the SW to behave erratically). Likewise, interoperation with human operators or other software systems can cause problems.

- In contrast to many hardware failures, which develop gradually (e.g., slowly worsening decrease in oil pressure due to a growing leak), most software failures occur instantaneously. The reason for this is that most software is discrete (state machines, decision logic) and failures occur, for example, by entering a wrong branch in the software, not properly handling an exception, or a division-by-zero. Proper modeling of software failures thus cannot directly use modeling techniques for hardware, as these typically rely on linear or at least continuous system behavior.
- Because many software systems interact with human operators, it is possible that the human can engage the system in an unexpected way due to the human misunderstanding the entire state of the system. In aeronautical applications on Flight Management Systems (FMS), this is called “mode confusion”. If a pilot experiences mode confusion, he or she can interact with the system in unanticipated ways. Although verification and validation (V&V) of the FMS software can anticipate many such interactions, it is possible that the human operator could give a command which is unanticipated for a given configuration of the FMS.
- Many hardware systems have highly complex physics-based models, which are used to predict the behavior of the system. These models can take the form of differential equations, difference equations, or other generative models based on physical laws. In general, software systems do not obey such physical laws. It is important to note, however, that the software systems themselves may be used to model or incorporate physical laws such as the case for motion dependent control systems. These models provide an interesting intermediate point between the two extremes of pure hardware systems with physical models and general software systems

All these differences (and commonalities) between ISHM of physical systems and software systems must be taken into account when developing novel techniques for intelligent software health management systems.

Some examples of major failures of safety-critical software systems can illustrate what kinds of software errors and problems an SWHM system has to cope with. In Ariane V, several software modules from the smaller Ariane IV had been re-used [32]. However, the range of certain sensor values was larger (due to different physical dimensions and construction), which led to an uncaught overflow error, causing the rocket to behave erratically and required its destruction.

The recent incident with the NASA DART probe [24] was mainly caused by software problems. One major issue was that the GPS receiver was replaced just prior to launch with a different model, with different noise and bias, without proper adaptation of the software. Automatic error correction in the navigation module caused wrong (biased) position and velocity values to be used as reference, causing the spacecraft to miss important trajectory points and to bump into the target satellite.

On December 3, 1999, a robotic spacecraft known as the Mars Polar Lander (MPL) was beginning descent into the Martian atmosphere when mission control lost all contact with the craft. An assessment [31] was made that spurious signals from the lander’s legs gave a false indication that the spacecraft had landed, resulting in a premature shut-down of the descent engines and subsequent crash on the surface. The interpretation of the signals from the lander legs was likely performed in software that had been rigorously tested. Although the software had been tested, it behaved unexpectedly due to signals that were unanticipated. This incident shows that complex software can react in unanticipated ways even after passing verification and validation testing.

Although we do not claim that a SWHM system could have prevented all of these software-related mishaps, it is possible that similar issues could be detected or avoided using health management techniques.

Like all fault detection and monitoring systems, our SWHM system will implemented as a piece of software itself. Safety analysis has to ask: “QUIS CUSTODIET IPSOS CUSTODES?” (Juvenal, “Who guards the guardians?”). This means that SWHM systems must be at least as safe and dependable as the software they monitor (“host software”).

2. SWHM GOALS

While the overall goal of SWHM is to extend and augment traditional V&V for a full lifecycle protection of software systems, thus ultimately enabling in-the-field assurance of composed software intensive systems, SWHM needs to provide the following capabilities:

- SWHM needs to *continuously monitor* the software under scrutiny. That software can be a compact piece of embedded software, or a huge, distributed software-rich system of systems, which might consist of heterogeneous components. It also must monitor the interactions of the software with the hardware, as many software faults originate (or are triggered) by anomalies in software-hardware interactions.
- SWHM should provide model-based fault detection, fault identification (root cause analysis) and decision support (for mitigation systems or human operators).
- SWHM should provide prognostic capabilities for enhanced system reliability, availability, and performance. SWHM will not only react on problems that already occurred, but should be able to give future prognosis on performance (e.g., by predicting computational bottlenecks), availability, and reliability (prognosis of looming problems, e.g., memory leaks, overfull file systems, overloaded network connections).
- SWHM will be capable of detecting environmental changes and *emerging behaviors*, as those cannot be detected (by definition) during pre-deployment verification and validation (V&V).
- SWHM will need to undergo rigorous V&V itself, as the SWHM must be at least as reliable as the system it monitors.
- SWHM models and reasoning capabilities must be able to minimize the number of false positives (spurious alarms) and false negatives (undetected failures).

- SWHM must be integrated seamlessly with traditional V&V, as it is not intended to replace V&V but to augment it for in-the-field software assurance. In particular, pre-deployment V&V will provide verification credit. Also V&V information will be perused to improve SWHM models.

3. SWHM TECHNOLOGY

The principal architecture of an SWHM system (Figure 1) consists of four components: the system (SW and hardware) to be monitored (the host system); a health model of the system; the SWHM reasoning engine that performs failure detection, diagnostics and prognostics; and components for failure annunciation or failure mitigation. In the architecture shown here, a key issue that arises is that the entire closed-loop system (including the SWHM executive) must meet stability and other control-oriented requirements and also be certifiable. The composite system which includes the hardware, SWHM executive, and aircraft controller is of higher complexity due to the addition of the SWHM executive and the additional requirements that it poses on the communication and computation environment. While the architecture shown here depicts the SWHM system as a single entity, in a real-world application it may itself be a distributed system taking information from multiple sources and performing reasoning on multiple computational platforms. This additional complexity can come with significant benefits. However, for real-world implementations, an overall cost-benefit analysis must be performed to justify the added complexity for flight-critical systems. In the event that each component of the architecture consists of certifiable components, it may be possible to demonstrate safety improvements and potentially cost reductions through the introduction of such a system.

While the SWHM architecture in Figure 1 specifically assumes an aircraft, the more general problems of monitoring the software of a cyber-physical system or an arbitrary software system is also of interest. Of course, the goals and requirements for different types of software systems vary, and for example safety, real-time, and V&V requirements are key in the aerospace setting.

Great progress has been made over the last decade, in learning and reasoning using probabilistic graphical models, including Bayesian networks (BN) and Markov networks. In addition to being well-suited to automated analysis, these graphical models are also amenable to visualization¹. While most BN inference problems are computationally hard in the general case, efficient algorithms have been developed [6, 17] that open the path toward successful applications in a wide range of automated reasoning areas, for example in model-based diagnosis (e.g., [21]), sensor validation [2, 22], or intelligent data analysis [15, 28].

In the following, we will focus on SWHM modeling and reasoning aspects, as their scalability is particularly important for large-scale and heterogeneous software systems. Under the assumption that modeling and reasoning is done using Bayesian networks and arithmetic circuits [6, 26], we will also briefly discuss existing techniques and research needed to improve V&V of SWHM.

¹e.g., <http://reasoning.cs.ucla.edu/samiam>

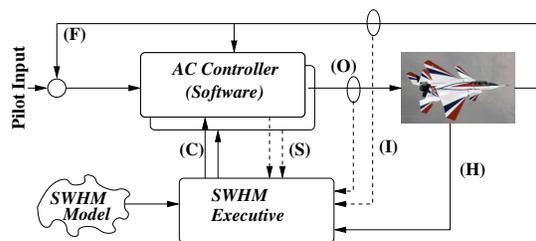


Figure 1: Principal architecture for SWHM for an aircraft controller: pilot input and feedback (F) produce actuator output (O). Hardware health sensors (H), signal quality data (I), and software quality data (S) go into the SWHM system, which produces a recovery/mitigation signal (C).

4. SWHM RESEARCH DIRECTIONS

Whereas ISHM is a mature field, research on the specific topic of software health management is still in its infancy. The 2009 SHM workshop [16], held during the Conference on Space Mission Challenges for Information Technology (SMC-IT 2009), gives an overview of some of the state-of-the-art approaches. Monitoring of software, while it is in operation, is obviously an important topic of research. Extending the notion of runtime monitors for runtime verification enables the designer to explore possible fault states of the software in advance (see [33, 8]). The dynamic monitoring of highly reliable and redundant software poses its own challenges (e.g., [12]). Other SWHM research focuses on specific software architectures that are particularly amenable for SWHM (e.g., [9]). Some research describes SWHM in architectures that conform to the ARINC 653 standard [1] while others discuss the automated generation of fault trees [18]. A process-oriented approach to regularly check on the health of a (large) software system has been discussed in [27]. Here, the goal is that regular (non-automated) health checks improve the technical condition of the software and has positive economic effectiveness.

In the rest of this section, and in the context of the SWHM architecture shown in Figure 1, we discuss important research directions that need to be addressed jointly by the software engineering and Bayesian network communities.

4.1 Advanced SWHM Modeling

Many software systems include a wide range of different and heterogeneous components along many dimensions, e.g., embedded vs. ground SW, autonomous vs. human-in-the-loop. As a consequence of the uncertain, heterogeneous, and interacting nature of these systems, as well as their environments, there is a need for supporting probabilistic modelling and analysis paradigms, techniques, and tools. Specifically, we emphasize large-scale probabilistic graphical models, in particular hierarchical and compositional Bayesian networks, and algorithms for probabilistic inference and machine learning using graphical models to ensure scalability to large software systems. These Bayesian graphical models allow the designer to specify large models (important for large software systems) in a hierarchical and structured way and use different levels of abstraction for the individual SW components. For example, Bayesian networks have been

used to diagnose problems in the operation of an operating system [4].

In this context, these are a few interesting research directions:

- Amount of sensing and sensor placement: where in the SW and HW stack should BN sensors be placed, how many are needed, and what are the opportunities for active diagnosis?
- Bayesian network construction: what is, for SWHM, the optimal approach to BN construction—auto-generation, machine learning, manual construction, or a hybrid approach?
- Higher-level modeling and analysis: what are the opportunities for developing and extending approaches to re-use of BNs, BN patterns, object-orientated BNs (OOBNs), and probabilistic model checking?

We believe that it is important to pursue research regarding, on the one hand, probabilistic modeling and reasoning approaches (including dynamic Bayesian networks), and on the other hand, probabilistic model checking and more generally formal methods that involve probabilities. In particular, there is a need to integrate dynamic Bayesian networks research and research on probabilistic model checking based on Markov chains, as previous research efforts have largely been pursued independently and in different research communities (Bayesian networks, software engineering, and formal methods).

4.2 SWHM with Arithmetic Circuits

Based upon sensor signals and health monitor signals, the SWHM engine tries to disambiguate the information and locate the failures that have occurred, using a representation of a model of the host software. The reasoning engine often also makes decisions on how to overcome the failure or to recover the system. Because most inference problems needed for fault detection and identification are computationally expensive, we propose to use techniques that *compile* BNs into a data-structure, which allows highly optimized and efficient processing. For Bayesian networks, the fourth author has developed a translation of BNs into arithmetic circuits [3, 5, 6]. Powerful optimizations keep these data structures compact, so that reasoning can be performed over large models. Future research will investigate the compilation of heterogeneous Bayesian network models. Additional interesting research questions that need to be addressed include:

- Interface between SWHM and host SW: what is the best way to structure this interaction—aspect-oriented techniques, message-passing, or something else?
- Mitigation: automated versus manual: should SWHM play a very active mitigation role or more of a supporting role for human analysis?
- Parametrization of compilation from BN to AC: given the wide range of existing and emerging SW and HW platforms (the emerging mobile-and-cloud architecture; varying computer architectures including multi-core CPUs and graphics processing units (GPUs)), how should one support the generation of widely varying ACs in terms of their memory and computational requirements?

4.3 V&V of SWHM Systems

A SWHM system as discussed above will be implemented as a piece of software. Based on our discussion above, it is obvious that the SWHM system is highly safety critical. False alarms or undetected faults can have severe consequences, ranging from unnecessary switching to redundant components to potential loss of life.

Therefore, all SWHM system components have to undergo rigorous V&V as well as certification. In general, this involves two major parts: (a) V&V of the SWHM model, i.e., assuring that the model reflects the software system and its failures/faults correctly and sufficiently, and (b) V&V of the algorithm and implementation of the proper SWHM system, i.e., the reasoning engine and executive that will be running during the operation of the overall system.

As discussed earlier, our SWHM models are represented as Bayesian networks, which can be compiled into arithmetic circuits of bounded size [6], which enables the SWHM engine to perform efficient reasoning. However, this executive is a highly non-standard algorithm, which means that specific V&V techniques are needed. In particular, the following research questions need to be addressed:

- Correctness and completeness of model compilation: will reasoning with arithmetic circuits always yield the same results as reasoning over the BN?
- Functional correctness of the ISHM reasoning executive: does the implementation of the executive perform the right kinds of reasoning operations on the compiled model?
- Runtime and memory limitations: can the run-time for reasoning be limited (real-time guarantee)? Can the memory requirements for reasoning be limited upfront, such that no dynamic memory handling is necessary?

For V&V, advanced verification and validation tools, like the Java PathFinder model checker², automatic generation of test-cases with symbolic PathFinder [25], compositional verification [11], and parametric testing [13, 29] will provide a basis. It is expected that techniques for the verification and validation of system health management software (e.g., [19, 30]) can be adapted to SWHM.

5. CONCLUSIONS

SWHM has the potential to become a key technology for detecting, diagnosing, predicting, and mitigating the adverse events during the operation of safety-critical software systems. The use of Bayesian networks for modeling and model-compilation into arithmetic circuits offers advantages (like well-defined semantics, wide range of techniques, tools, and algorithms, as well as exact compilation), which makes their use in monitoring the health of safety-critical and embedded software possible. Approaches for V&V and certification of SWHM can be based upon advanced verification tools (like model checking), but substantial research will be necessary to address these issues.

Acknowledgements This work is in part supported by the NASA Aviation Safety Program IVHM project (NRA NNX08AY50A).

²<http://javapathfinder.sourceforge.net>

6. REFERENCES

- [1] M. Barry and G. Horvath. Goal-based Flight Software Health Management Services. In *SHM 2009* [16], 2009.
- [2] T. W. Bickmore. A probabilistic Approach to Sensor Data Validation. In *AIAA, SAE, ASME, and ASEE 28th Joint Propulsion Conf. and Exhibit*, 1992.
- [3] M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *Proc. IJCAI-07*, pages 2443–2449, 2007.
- [4] G. Cooper, D. Heckerman, and C. Meek. A Bayesian Approach to Causal Discovery. Microsoft Research Technical Report MSR-TR-97-05, 1997.
- [5] A. Darwiche. A differential Approach to Inference in Bayesian Networks. *JACM*, 50(3):280–305, 2003.
- [6] A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.
- [7] M. desJardins, P. Rathod, and L. Getoor. Bayesian Network Learning with Abstraction Hierarchies and context-specific Independence. In *Proc. ECML-2005*, volume 16, 2005.
- [8] W. Dong, M. Leucker, and C. Schallhart. Impartial anticipations in runtime verification. In *6th Int. Symp. on Automated Technology for Verification and Analysis (ATVA '08)*, vol 5311 LNCS. Springer, 2008.
- [9] A. Dubey, G. Karsai, R. Kereskenyi, and M. Mahadevan. A Real-Time Component Framework: Experience with CCM and ARINC-653. *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 2010.
- [10] N. Friedman and M. Goldszmidt. Learning Bayesian Networks with local Structure. In *Proc. UAI-96*, pages 252–262, 1996.
- [11] D. Giannakopoulou and C. S. Pasareanu. Interface Generation and compositional Verification in Java PathFinder. In *FASE*, vol 5503 of LNCS, pages 94–108. Springer, 2009.
- [12] A. Goodlow and L. Pike. Toward Monitoring fault-tolerant embedded Systems. In *SHM 2009* [16], 2009.
- [13] K. Gundy-Burlet, J. Schumann, T. Menzies, and T. Barrett. Parametric Analysis of ANTARES Re-entry Guidance Algorithms using advanced Test Generation and Data Analysis. In *iSAIRAS*, 2008.
- [14] Y. Guo, D. Wilkinson, and D. Schuurmans. Maximum Margin Bayesian Networks. In *Proc. UAI*, page 233, 2005. AUAI Press.
- [15] P. Jones, C. Hayes, D. Wilkins, R. Bargar, J. Sniezek, P. Asaro, O. J. Mengshoel, D. Kessler, M. Lucenti, I. Choi, N. Tu, and J. Schlabach. CoRAVEN: Modeling and Design of a Multimedia intelligent Infrastructure for collaborative Intelligence Analysis. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*, pages 914–919, 1998.
- [16] G. Karsai, editor. *1st International Workshop on Software Health Management (SHM 2009)*. ISIS, Vanderbilt University, 2009.
- [17] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles And Techniques*. MIT Press, 2009.
- [18] T. Kurtoglu, R. Lutz, and A. Patterson-Hine. Using auto-generated Diagnostic Trees for optimized Fault Handling. In *SHM 2009* [16], 2009.
- [19] A. E. Lindsey and C. Pecheur. Simulation-based Verification of autonomous Controllers via Livingstone Pathfinder. In *TACAS 2004*, vol 2988 of LNCS, pages 357–371. Springer, 2004.
- [20] O. J. Mengshoel. Understanding the Role of Noise in Stochastic Local Search: Analysis and Experiments. *Artificial Intelligence* 172:8–9, pages 955–990, 2008.
- [21] O. J. Mengshoel, A. Darwiche, K. Cascio, M. Chavira, S. Poll, and S. Uckun. Diagnosing Faults in electrical Power Systems of Spacecraft and Aircraft. In *Proc. IAAI-08*, pages 1699–1705, 2008.
- [22] O. J. Mengshoel, A. Darwiche, and S. Uckun. Sensor Validation using Bayesian Networks. In *iSAIRAS*, 2008.
- [23] O. J. Mengshoel, D. Roth, and D. C. Wilkins. Portfolios in Stochastic Local Search: Efficiently Computing Most Probable Explanations in Bayesian Networks. *JAR* (accepted), 2010.
- [24] NASA. Overview of the DART Mishap Investigation Results. http://www.nasa.gov/pdf/148072main_DART_mishap_overview.pdf, 2006.
- [25] C. S. Pasareanu and W. Visser. Symbolic Execution and Model Checking for Testing. In *Haifa Verification Conf.*, vol 4899 of LNCS, pages 17–18. Springer, 2007.
- [26] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.
- [27] M. Pizka and T. Panas. Establishing economic Effectiveness through Software Health Management. In *SHM 2009* [16], 2009.
- [28] C. C. Ruokangas and O. J. Mengshoel. Information filtering using Bayesian networks: Effective user interfaces for Aviation Weather Data. In *Int. Conference on intelligent User Interfaces*, pages 280–283, 2003.
- [29] J. Schumann, A. Bajwa, and P. Berg. Parametric Testing of Launch Vehicle FDDR Models. In *AIAA Space*, 2010.
- [30] J. Schumann, A. Srivastava, and O. Mengshoel. Who guards the Guardians? — toward V&V of Health Management Software (short paper). In *Runtime Verification 2010*, vol 5418 LNCS. Springer, 2010.
- [31] P. Willhide. Mars Program Assessment Report Outlines Route to Success, <http://mars.jpl.nasa.gov/msp98/news/news71>, 2000.
- [32] Wired.com: "History's Worst Software Bugs", 2009.
- [33] C. Zhao, W. Dong, J. Wang, P. Sui, and Z. Qi. Software active online Monitoring under anticipatory Semantics. In *SHM 2009*, 2009.