

BAYESIAN SOFTWARE HEALTH MANAGEMENT FOR AIRCRAFT GUIDANCE, NAVIGATION, AND CONTROL

Johann Schumann¹, Timmy Mbaya², Ole Mengshoel³

¹ *SGT, Inc. NASA Ames, Moffett Field, CA 94035*
Johann.M.Schumann@nasa.gov

² *University of Massachusetts, Boston*
timstim@mail.com

³ *Carnegie Mellon University, NASA Ames, Moffett Field, CA 94035*
Ole.Mengshoel@sv.cmu.edu

ABSTRACT

Modern aircraft—both piloted fly-by-wire commercial aircraft as well as UAVs—more and more depend on highly complex safety critical software systems with many sensors and computer-controlled actuators. Despite careful design and V&V of the software, severe incidents have happened due to malfunctioning software.

In this paper, we discuss the use of Bayesian networks to monitor the health of the on-board software and sensor system, and to perform advanced on-board diagnostic reasoning. We focus on the development of reliable and robust health models for combined software and sensor systems, with application to guidance, navigation, and control (GN&C). Our Bayesian network-based approach is illustrated for a simplified GN&C system implemented using the open source real-time operating system OSEK/Trampoline. We show, using scenarios with injected faults, that our approach is able to detect and diagnose faults in software and sensor systems.

1. INTRODUCTION

Modern aircraft depend increasingly on the reliable operation of complex, yet highly safety-critical software systems. Fly-by-wire commercial aircraft and UAVs are fully controlled by software. Failures in the software or a problematic software-hardware interaction can have disastrous consequences.

Although on-board diagnostic systems nowadays exist for most aircraft (hardware) subsystems, they are mainly working independently from each other and are not capable of reliably determining the root cause or causes of failures, in particular when software failures are to blame. Clearly, a powerful FDIR (Fault Detection, Isolation, Recovery) or ISHM

(Integrated System Health Management) system for *software* has a great potential for ensuring safety and operational reliability of aircraft and UAVs. This is particularly true, since many software problems do not directly manifest themselves but rather exhibit *emergent behavior*. For example, when the F-22 Raptors crossed the international date line, a software problem in the guidance, navigation, and control (GN&C) system did not only shut down that safety-critical component but also brought down communications, so the F-22s had to be guided back to Hawaii using visual flight rules.¹

An on-board software health management (SWHM) system monitors the flight-critical software while it is in operation, and thus is able to detect faults, such as the F-22 problems, as soon as they occur. In particular, an SWHM system

- *monitors the behavior of the software and interacting hardware during system operation.* Information about operational status, signal quality, quality of computation, reported errors, etc., is collected and processed on-board. Since many software faults are caused by problematic hardware/software interactions, status information about software components must be collected and processed, in addition to that for hardware.
- *performs diagnostic reasoning in order to identify the most likely root cause(s) for the fault(s).* This diagnostic capability is extremely important. In particular, for UAVs, the available bandwidth for telemetry is severely limited; a “dump” of the system state and analysis by the ground crew in case of a problem is not possible.

For manned aircraft, an SWHM can reduce the pilot’s workload substantially. With a traditional on-board diagnostic system, the pilot can get swamped by diagnostic errors and warnings coming from many different subsystems. Recently, when one of the engines exploded on a

Johann Schumann et.al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 United States License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

¹<http://www.af.mil/news/story.asp?storyID=123041567>

Qantas A380, the pilot had to sort through literally hundreds of diagnostic messages in order to find out what happened. In addition, several diagnostic messages contradicted each other.²

In this paper, we describe our approach of using Bayesian networks as the modeling and reasoning paradigm to achieve SWHM. With a properly developed Bayesian network, detection of faults and reasoning about root causes can be performed in a principled way. Also, a proper probabilistic treatment of the diagnosis process, as we accomplish with our Bayesian approach (Pearl, 1988; Darwiche, 2009), can not only merge information from multiple sources but also provide a posterior distribution for the diagnosis and thus provide a metric for the quality of this result. We note that this approach has been very successful for electrical power system diagnosis (Ricks & Mengshoel, 2009, 2010; Mengshoel et al., 2010).

It is obvious that an SWHM system that is supposed to operate on-board an aircraft, in an embedded environment, must satisfy important properties: first, the implementation of the SWHM must have a small memory and computational footprint and must be certifiable. Second, the SWHM should exhibit a low number of false positives and false negatives. False alarms (false positives) can produce nuisance signals; missed adverse events (false negatives) can be a safety hazard. Our approach of using SWHM models, that have been compiled into arithmetic circuits, are amenable to V&V (Schumann, Mengshoel, & Mbaya, 2011).

The remainder of the paper is structured as follows: Section 2. introduces Bayesian networks and how they can be used for general diagnostics. In Section 3. we demonstrate our approach to software health management with Bayesian networks and discuss how Bayesian SWHM models can be constructed. Section 4. illustrates our SHWM approach with a detailed example. We briefly describe the demonstration architecture and the example scenario, discuss the use of a Bayesian health model to diagnose such scenarios, and present simulation results. Finally, in Section 5. we conclude and identify future work.

2. BAYESIAN NETWORKS

Bayesian networks (BNs) represent multivariate probability distributions and are used for reasoning and learning under uncertainty (Pearl, 1988). They are often used to model systems of a (partly) probabilistic nature. Roughly speaking, random variables are represented as nodes in a directed acyclic graph (DAG), while conditional dependencies between variables are represented as graph edges (see Figure 1 for an example). A key point is that a BN, whose graph structure often

reflects a domain’s causal structure, is a compact representation of a joint probability table if the DAG is relatively sparse. In a discrete BN (as we are using for SWHM), each random variable (or node) has a finite number of states and is parameterized by a conditional probability table (CPT).

During system operation, observations about the software and system (e.g., monitoring signals and commands) are mapped into states of nodes in the BN. Various probabilistic queries can be formulated based on the assertion of these observations to yield predictions or diagnoses for the system. Common BN queries of interest include computing posterior probabilities and finding the most probable explanation (MPE). For example, an observation about abnormal behavior of a software component could, by computing the MPE, be used to identify one or more components that are most likely in faulty states.

Different BN inference algorithms can be used to answer the queries. These algorithms include join tree propagation (Lauritzen & Spiegelhalter, 1988; Jensen, Lauritzen, & Olesen, 1990; Shenoy, 1989), conditioning (Darwiche, 2001), variable elimination (Li & D’Ambrosio, 1994; Zhang & Poole, 1996), and arithmetic circuit evaluation (Darwiche, 2003; Chavira & Darwiche, 2007). In resource-bounded systems, including real-time avionics systems, there is a strong need to align the resource consumption of diagnostic computation with resource bounds (Musliner et al., 1995; Mengshoel, 2007) while also providing predictable real-time performance. The compilation approach—which includes join tree propagation and arithmetic circuit evaluation—is attractive in such resource-bounded systems.

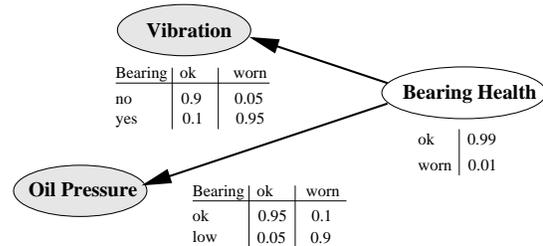


Figure 1. Simple Bayesian network. CPT tables are shown near each node.

Let us consider a very simple example of a Bayesian network (Figure 1) as it could be used in diagnostics. We have a node Bearing Health (*BH*) representing the health of a ball bearing in a diesel engine, a sensor node Vibration (*V*) representing whether vibration is measured or not, and a node Oil Pressure (*OP*) representing oil pressure. Clearly, the sensor readings depend on the health status of the ball bearing, and this is reflected by the directed edges. The degrees of influence are defined in the two CPTs depicted next to the sensor nodes. For example, if there is vibration, the probability that $p(BH = \sim \text{ok})$ increases. To obtain the health of the ball bearing, we input (or clamp) the states of the BN

²<http://www.aerosocietychannel.com/aerospace-insight/2010/12/exclusive-qantas-qf32-flight-from-the-cockpit/>

sensor nodes and compute the posterior distribution (or belief) over BH . The prior distribution of failure, as reflected in the CPT shown next to BH , is also taken into account in this calculation.

Our example network in Figure 1 represents the joint probability $p(BH, V, OP)$ and is shown in Table 1. For simplicity, we replace all CPT entries with θ_x (i.e., $\theta_{ok} \leftrightarrow BH$ is ok, and $\theta_{\sim ok} \leftrightarrow BH$ is worn). Let λ_i indicate whether evidence of a specific state is observed (i.e., $\lambda_v = 1$ means evidence of vibration is observed, and $\lambda_v = 0$ means no evidence of vibration is observed). The probability distribution $p(BH, V, OP)$ captured by the Bayesian network above is shown in Table 1.

| BH | V | OP | $p(BH, V, OP)$ |
|-----------|----------|-----------|--|
| ok | v | op | $\lambda_{ok} \lambda_v \lambda_{op} \theta_{v ok} \theta_{ok} \theta_{op ok}$ |
| ok | v | \sim op | $\lambda_{ok} \lambda_v \lambda_{\sim op} \theta_{v ok} \theta_{ok} \theta_{\sim op ok}$ |
| ok | \sim v | \sim op | $\lambda_{ok} \lambda_{\sim v} \lambda_{\sim op} \theta_{\sim v ok} \theta_{ok} \theta_{\sim op ok}$ |
| ok | \sim v | op | $\lambda_{ok} \lambda_{\sim v} \lambda_{op} \theta_{\sim v ok} \theta_{ok} \theta_{op ok}$ |
| \sim ok | v | op | $\lambda_{\sim ok} \lambda_v \lambda_{op} \theta_{v \sim ok} \theta_{\sim ok} \theta_{op \sim ok}$ |
| \sim ok | \sim v | op | $\lambda_{\sim ok} \lambda_{\sim v} \lambda_{op} \theta_{\sim v \sim ok} \theta_{\sim ok} \theta_{op \sim ok}$ |
| \sim ok | v | \sim op | $\lambda_{\sim ok} \lambda_v \lambda_{\sim op} \theta_{v \sim ok} \theta_{\sim ok} \theta_{\sim op \sim ok}$ |
| \sim ok | \sim v | \sim op | $\lambda_{\sim ok} \lambda_{\sim v} \lambda_{\sim op} \theta_{\sim v \sim ok} \theta_{\sim ok} \theta_{\sim op \sim ok}$ |

Table 1. Probability distribution for $p(BH, V, OP)$.

According to this joint probability distribution table, the first row ($\lambda_{ok} \lambda_v \lambda_{op} \theta_{v|ok} \theta_{ok} \theta_{op|ok}$) is representing the probability that the health of the ball bearing is okay ($\lambda_{ok} = 1$), and that vibrations and good oil pressure are observed (λ_v and $\lambda_{op} = 1$) would be 9.4% indicating a very low degree of belief in such a state. Given the corresponding numerical CPT entries this number is calculated as $\theta_{v|ok} \theta_{ok} \theta_{op|ok} = 0.1 * 0.99 * 0.95 = 0.09405$. On the other hand, the fourth row ($\lambda_{ok} \lambda_{\sim v} \lambda_{op} \theta_{\sim v|ok} \theta_{ok} \theta_{op|ok}$) representing the probability that the ball bearing is okay ($\lambda_{ok} = 1$), there is no vibrations and good oil pressure ($\lambda_{\sim v}$ and $\lambda_{op} = 1$) is much higher (85%) as follows: $\theta_{\sim v|ok} \theta_{ok} \theta_{op|ok} = 0.9 * 0.99 * 0.95 = 0.84645$.

Posterior marginals can be computed from the joint distribution:

$$p(BH, V, OP) = \prod_{\theta_{s|x}} \prod_{\lambda_s} \lambda_s$$

where $\theta_{s|x}$ indicates a state's conditional probability and λ_s indicates whether or not state s is observed. Here, θ variables are known as variables, λ variables as indicators.

Summing all individual joint distribution entries yields a multi-linear function—at the core of arithmetic circuit evaluation—referred to as the *network polynomial* f

(Darwiche, 2009):

$$f = \lambda_{ok} \lambda_v \lambda_{op} \theta_{v|ok} \theta_{ok} \theta_{op|ok} + \lambda_{ok} \lambda_v \lambda_{\sim op} \theta_{v|ok} \theta_{ok} \theta_{\sim op|ok} + \lambda_{ok} \lambda_{\sim v} \lambda_{\sim op} \theta_{\sim v|ok} \theta_{ok} \theta_{\sim op|ok} + \lambda_{ok} \lambda_{\sim v} \lambda_{op} \theta_{\sim v|ok} \theta_{ok} \theta_{op|ok} + \lambda_{\sim ok} \lambda_v \lambda_{op} \theta_{v|\sim ok} \theta_{\sim ok} \theta_{op|\sim ok} + \lambda_{\sim ok} \lambda_{\sim v} \lambda_{op} \theta_{\sim v|\sim ok} \theta_{\sim ok} \theta_{op|\sim ok} + \lambda_{\sim ok} \lambda_v \lambda_{\sim op} \theta_{v|\sim ok} \theta_{\sim ok} \theta_{\sim op|\sim ok} + \lambda_{\sim ok} \lambda_{\sim v} \lambda_{\sim op} \theta_{\sim v|\sim ok} \theta_{\sim ok} \theta_{\sim op|\sim ok},$$

or in other words

$$f = \sum_E \prod_{\theta_{s|x}} \prod_{\lambda_s} \lambda_s$$

where E indicates evidence of a network instantiation.

An arithmetic is a compact representation of a network polynomial. An arithmetic circuit (AC) is a directed acyclic graph (DAG) in which leaf nodes represent variables (parameters and indicators) while other nodes represent addition and multiplication operators. Size, in terms of number of AC edges, is a measure of complexity of inference. Unlike treewidth, another complexity measure, AC size can take network parameters (such as determinism and local structure) into account.

Answers to probabilistic queries, including marginals and MPE, are computed using algorithms that operate directly on the arithmetic circuit. A bottom-up pass over the circuit, from input to output, evaluates the probability of a particular evidence setting (or clamping of λ parameters) on the state of the network. And a top-down pass over the circuit, from output to input, computes partial derivatives. From these partial derivatives one can compute many marginal probabilities, provide information about how change in a specific node affects the whole network (sensitivity analysis), and perform MPE computation (Darwiche, 2009).

3. BAYESIAN NETWORKS FOR SOFTWARE HEALTH MANAGEMENT

At a first glance, the SWHM does look very similar to a traditional integrated vehicle health management system (IVHM): sensor signals are interpreted to detect and identify any faults, which are then reported. Such FDIR systems are nowadays commonplace in the aircraft and for other complex machinery. It seems like it would be straight-forward to attach a software to be monitored (host software) to such an FDIR. However, there are several critical differences between FDIR for hardware and software health management. Most prominently, many software faults do not develop gradually over time (e.g., like an oil leak); rather they occur instantaneously. Whereas some of the software faults directly impact the current software module (e.g., when a division-by-zero is detected), there are situations where the effects of a software fault manifest themselves in an entirely different subsystem, as discussed in the F-22 example above. For this reason, and

the fact that many software problems occur due to problematic SW/HW interactions, both software and hardware must be monitored in an integrated fashion.

Based upon requirements as laid out in Section 1., we are using Bayesian networks to develop SWHM models. On a top-level, data from software and hardware sensors are presented to the nodes of the Bayesian network, which in turn performs its reasoning (i.e., updating the internal health and status nodes) and returns information about the health of the software (or specific components thereof). The information about the health of the software is extracted from the posterior distribution, specifically from health nodes. In our modeling approach, we chose to use Bayesian networks, which do not reason about temporal sequences (i.e., dynamic Bayesian networks) because of their complexity. Therefore, all sensor data, which are usually time series, must undergo a pre-processing step, where certain (scalar) *features* are extracted. These values are then discretized into symbolic states (e.g., “low”, “high”) or normalized numeric values before presented to the Bayesian health model (Section 3.3).

3.1 Bayesian SWHM

3.1.1 Nodes

Our Bayesian SWHM models are set up using several kinds of nodes. Please note that all nodes are discrete, i.e., each node has a finite number of mutually exclusive and exhaustive states.

CMD node C Signals sent to these nodes are handled as ground truth and are used to indicate commands, actions, modes or other (known) states. For example, a node `Write_File_System` represents an action, which eventually will write some data into the file system, has been commanded. For our reasoning it is assumed that this action is in fact happening.³ The CMD nodes are root nodes (no incoming edges). During the execution of the SWHM, these nodes are always directly connected (clamped) to the appropriate command signals.

SENSOR node S A sensor node S is an input node similar to the CMD node. The data fed into this node are sensor data, i.e., measurements that have been obtained from monitoring the software or the hardware. Thus, this signal is not necessarily correct. It can be noisy or wrong altogether. Therefore, a sensor node is typically connected with a health node, that describes the health status of the sensor node.

HEALTH node H The health nodes are nodes that reflect the health status of a sensor or component. Their posterior probabilities comprise the output of an SWHM

³If there is a reason that this command signal is not reliable, the command node C is used in combination with a H node to impact state U as further discussed below. Alternatively, one might consider using a sensor node instead.

model. A health node can be binary (with states, say, `ok` or `bad`), or can have more states that reflect health status at a more fine-grained level. Health nodes are usually connected to sensor and status nodes.

STATUS node U A status node reflects the (unobservable) status of the software component or subsystem.

BEHAVIOR node B Behavior nodes connect sensor, command, and status nodes and are used to recognize certain behavioral patterns. The status of these nodes is also unobservable, similar to the status nodes. However, usually no health node is attached to the behavioral nodes.

3.1.2 Edges

The following informal way to think about edges in Bayesian networks are useful for knowledge engineering purposes: An edge (arrow) from node C to node E indicates that the state of C has a (causal) influence on the state of E .

Suppose that S is a software signal (e.g., within the aircraft controller) that leads into an input port I of the controller. Let us assume that we want S being 1 to cause C to be 1 as well. Failure mechanisms are represented by introduced a health node H . In our example, we would introduce a node H and let it be a (second) parent of I . More generally, the types of influences typically seen in the SWHM BNs are as follows:

$\{H, C\} \rightarrow U$ represents how state U may be commanded through command C , which may not always work as indicated. This is reflected by the health H of the command mechanism’s influence on the state.

$\{C\} \rightarrow U$ represents how state U may be changed through command C ; the health of the command mechanism is not explicitly represented. Instead, imperfections in the command mechanism can be represented in the CPT of U .

$\{H, U\} \rightarrow S$ represents the influence of system status U on a sensor S , which may also fail as reflected in H . We use a sensor to better understand what is happening in a system. However, the sensor might give noisy readings; the level of noise is reflected in the CPT of S .

$\{H\} \rightarrow S$ represents a direct influence of system health H on a sensor S , without modeling of state (as is done in the $\{H, U\} \rightarrow S$ pattern). An example of this approach is given in Figure 1.

$\{U\} \rightarrow S$ represents how system status U influences a sensor S . Sensor noise and failure can both be rolled into the CPT of S .

Table 2 shows the CPT for the last case. Here, we consider the status of a file system (FS). The file system can be `empty`, `full`, or filled to more than 95% (`full95`). If more space is available, its state is labeled `ok`. This (unobservable) state is observed by a software sensor, which measures the current

capacity of the file system (FC). Because this sensor might fail, a health node (FH) indicates the health of FC sensor as ok or bad.

Because the sensor node FC has two parents (status node FS and health node FH), the CPT table is 3-dimensional. Table 2 flattens out this information: the rows correspond to the states of the sensor node (1st group for healthy sensor, 2nd group for bad sensor). The rightmost four columns refer to the states of the FS node. In this particular example, a file system sensor, which is not working properly will not report if the file system is almost full or full. Such a bad sensor will only report empty or ok. This is reflected by the zero-entries in the lower right corner of the CPT.

| FS | FH | $p(FC FH, FS)$ | | | |
|---------|------|----------------|------|---------|------|
| | | empty | ok | full195 | full |
| empty | ok | 0.88 | 0.05 | 0.01 | 0.01 |
| ok | ok | 0.1 | 0.6 | 0.2 | 0.1 |
| full195 | ok | 0 | 0.2 | 0.7 | 0.1 |
| full | ok | 0 | 0 | 0 | 1 |
| empty | bad | 0.9 | 0.1 | 0 | 0 |
| ok | bad | 0.1 | 0.9 | 0 | 0 |
| full195 | bad | 0.5 | 0.5 | 0 | 0 |
| full | bad | 0.5 | 0.5 | 0 | 0 |

Table 2. CPT table for $p(FC|FH, FS)$.

3.1.3 Developing Conditional Probability Tables (CPTs)

The CPT entries are set based on a priori and empirical knowledge of a system's components and their interactions (Ricks & Mengshoel, 2009; Mengshoel et al., 2010). This knowledge may come from different sources, including (but not restricted to) system schematics, source code, analysis of prior software failures, and system testing. As far as a system's individual components, mean-time-to-failure statistics are known for certain hardware components, however similar statistics are well-established for software. Consequently, further research is needed to determine the prior distribution for health states, including bugs, for a broad range of software components. As far as an interaction between a system's components, CPT entries can also be obtained from understanding component interactions, a priori, or testing how different components impact each other. As an example, consider a testbed like NASA's advanced diagnostic and prognostic testbed (ADAPT) (Poll et al., 2007), which provides both schematics and testing opportunities. Using a testing approach, one may inject specific states into the navigation system and record the impact on states of the guidance system, and perform statistical analysis, in order to guide the development of CPT entries for the guidance system. Setting of software component CPTs to reflect their interactions with hardware can be conducted in a similar way. Clearly, the well-known limitation of brute-force testing apply, and when this

occurs one needs to utilize design artifacts, system schematics, source code, and other sources of knowledge about component interactions.

3.2 Software Sensors

Information that is needed to reason about software health must be extracted from the software itself and all components that interact with the software, i.e., hardware sensors, actuators, the operating system, middleware, and the computer hardware. Different software sensors provide information about the software on different levels of granularity and abstraction. Table 3 gives an impression of the various layers of information extraction.

Only if information is available on different levels, the SWHM gets a reasonably complete picture of the current situation, which is an enabling factor for fault detection and identification. Information directly extracted from the software (Table 3) provide very detailed and timely information. However, this information might not be sufficient to identify a failure. For example, the aircraft control task might be working properly (i.e., no faults show up from the software sensors). However, some other task might consume too many resources (e.g., CPU time, memory, etc.), which in turn can lead to failures related to the control task. We therefore extract a multitude of different, usually readily available information about the software.

| Software | |
|------------------|----------------------------------|
| errors | flagged errors and exceptions |
| memsize | used memory |
| quality | signal quality |
| reset | filter reset (for navigation) |
| Software Intent | |
| fs_write | intent to write to FS |
| fork | intent to create new process(es) |
| malloc | intent to allocate memory |
| use_msg | intent to use message queues |
| use_sem | using semaphores |
| use_recursion | using recursion |
| Operating system | |
| cpu | CPU load |
| n_proc | number of processes |
| m_free | available memory |
| d_free | percentage of free disk space |
| shm | size of available shared memory |
| sema | information about semaphores |
| realtime | missed deadlines |
| n_intr | number of interrupts |
| l_msgqueue | length of message queues |

Table 3. SWHM informations sources

3.3 Preprocessing of Software and Hardware Sensor Data

The main goals of preprocessing are to extract important information from the (large amounts of) temporal sensor data and to discretize continuous sensor data to be used with our discrete SWHM models. For example, the sensor for the file system (*FC*) has the states *empty*, *ok*, *full195*, *full*. Preprocessing steps, which extract temporal features from raw sensor data, enable us to perform temporal reasoning without having to use a dynamic Bayesian network (DBN). This is a very prominent conceptual decision. By giving up the ability to do full temporal reasoning by means of DBNs, which may be complex in design and execution, we are able to use much simpler static health models and handle the temporal aspects during preprocessing.

In particular, we use the following preprocessing techniques (which can also be combined):

discretization A continuous value is discretized using a number of monotonically increasing thresholds. For example, Table 4 shows the discretization for file system sensor *FC*.

min/max/average The minimal/maximal value or the mean of the entire available time series is taken.

moving min/max/average A moving min/max/mean value (with a selectable window size) is taken. In contrast to the features above, we only consider the last few seconds of the signal.

sum (integral) The sum (integral) of the sensor value is taken. For example, the sum of “bytes-written-to-file-system” (per time unit) approximates the amount of data in the file system (assuming nothing is being deleted).

temporal Temporal states of sensor signals can be extracted, e.g., time difference between event *A* and event *B*.

time-series analysis Kalman filters can be used to correlate signals against a model. Residual errors then can be used as sensor states (e.g., close-to-model, small-deviation, large-deviation). Fast Fourier transformation (FFT) can be used to detect cyclic events, e.g., vibrations or oscillations.

| Percentage (df) | State |
|---------------------|----------------|
| $0 \leq df < 5\%$ | <i>empty</i> |
| $5 \leq df < 80\%$ | <i>ok</i> |
| $80 \leq df < 95\%$ | <i>full195</i> |
| $95 \leq df$ | <i>full</i> |

Table 4. Discretization into states (right) by means of thresholds (left).

4. DEMONSTRATION EXAMPLE

4.1 System Architecture

For demonstration purposes, we have implemented a simple system architecture on a platform that reflects real-time embedded execution typical of aircraft and satellite systems. Trampoline,⁴ an emulator for the OSEK⁵ real-time operating system (RTOS), is used as a platform rather than other RTOSes more established in the aerospace industry (such as Wind River’s VxWorks or GreenHills’ INTEGRITY). OSEK is easily available, widely used for embedded control systems in the automotive industry, and its capabilities were sufficient for the purpose of our experiments.

The basic system architecture (Figure 2) for running SWHM experiments consists of the OSEK RTOS, which runs a number of tasks or processes at a fixed schedule. For this simple SWHM demonstration system, (1) the simulation model of the plant is integrated as one of the OSEK tasks, and (2) hardware actuators and sensors are not modelled in detail, which would have required drivers and interrupts routines. Despite its simplicity, this architecture is sufficient to run a simple simulation of the aircraft and the GN&C software in a real-time environment (fixed time slots, fixed memory, inter-process communication, shared resources).

The software health management executive, including preprocessing, is executed as a separate OSEK task. It reads software and sensor data, performs preprocessing and provides the data as evidence to the sensor nodes of the (compiled) Bayesian network. The reasoning process then yields the posterior probabilities of the health nodes.

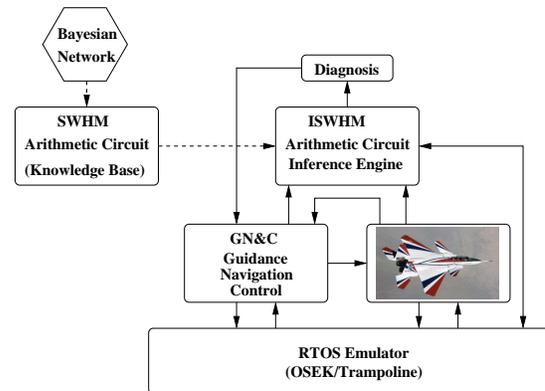


Figure 2. Demonstration system architecture. The Bayesian network model is compiled (before deployment) into an arithmetic circuit representing the knowledge base. The real-time operating system schedules three tasks: the controller, the plant, and the SWHM inference engine.

⁴<http://trampoline.rts-software.org/>

⁵<http://www.osek-vdx.org/>

4.2 Example Scenario

An experimental scenario aimed at the study of faults related to file systems, inspired by the Mars rover SPIRIT reboot cycle incident (Adler, 2006), has been implemented using the system architecture. A short time after landing, the Mars rover SPIRIT encountered repeated reboots, because a fault during the booting process caused a subsequent reboot. According to reports (Adler, 2006), an on-board file system for intermediate data storage caused the problem. After this storage was filled up, the boot process failed while trying to access that file system. The problem could be detected on the ground and was resolved successfully.

In a more general setting, this scenario is dealing with bad interaction due to scarce resources, and delays during access. Even if no errors show up, a blocking write access to a file system that is almost full, or the delivery of a message through a lengthy message queue, can in the worst case cause severe problems and emerging behavior.

For the purpose of demonstration, we designed a flawed software architecture with a global message queue that buffers all control signals and logs them in the file system (blocking) before forwarding them (Figure 3). This message queue is also used to transport image data from an on-board camera (e.g., for UAV) to the radio transmitter. The relevant software components of this simple architecture are: GN&C, message queue, logging to file system, camera, transmitter, and plant. On-board camera and transmitter are shown in Figure 3 but not used in the experiments described in this paper.

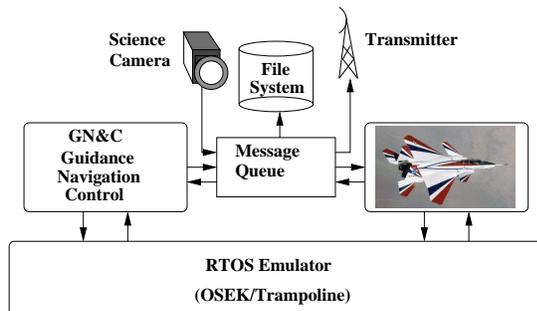


Figure 3. Software architecture for file system related fault scenarios, diagnosed using SWHM system.

Here, we are running the following scenario: the file system is initially set to almost full. Subsequent control messages, which are being logged, might stay longer in the message queue, because the blocking write into an almost full file system takes substantial time. This situation potentially causes overflow of the message queue or leads to loss of messages. However, even a small delay (i.e., a control message is not processed within its allotted time frame, but one or more time-frames later) can cause *oscillation* of the entire aircraft. This oscillation, similar to PIO (pilot induced oscillation) can lead

to dangerous situations or even loss of the aircraft.

In this scenario, the software problem does not manifest itself within the software system (e.g., in form of errors or exceptions). Rather, the overall behavior of the aircraft is impacted in a non-obvious way.

Other possible scenarios with this setup, to be diagnosed by the SWHM task, are:

- The pilot's or autopilot's stick commands are delayed, which again results in oscillations of the aircraft.
- Non-matching I/O signal transmit/read/processing rates between control stick and actuators result in plant oscillations whose root causes are to be disambiguated.
- An unexpectedly large feed from the on-board camera (potentially combined with a temporary low transmission bandwidth) can cause the message queue to overflow, which results in delays or dropped messages with similar effects as discussed above.
- The controller and the science camera compete for the message queue, which could (when not implemented correctly) cause message drops or even deadlocks.

With our SWHM, the observed problem (oscillation) should be detected properly and traced back to the root cause(s).

4.3 The SWHM Model

A Bayesian SWHM model for this architecture was designed using the SamIam tool.⁶ A modular BN design approach was attempted by first designing the SWHM model for the basic system including relevant nodes such as—in the aircraft case—the pitch-up and pitch-down command nodes. The pitch status nodes, the fuel status node, and the software, pitch, and acceleration health nodes were introduced. Other subnetworks were then added to this core Bayesian network to obtain the complete SHWM model for the specific architecture used for SWHM experiments. The relevant nodes of the subnetwork module added for SWHM experiment with file system related faults are shown in Figure 4.

The `Write_File_System` command node indicates whether a write to the file system is being executed. The health nodes for the file system and the message queue reflect the probabilities that they might malfunction. The status nodes for the file system and the message queue represent their unobservable states, while their sensor nodes reflect sensor readings after preprocessing.

The only non-standard software sensor node in this SWHM model is a sensor to detect oscillations or vibrations. A fast Fourier transform (FFT) performs a simple time-series analysis on major aircraft signals (e.g., accelerations or pqr rates). With such a sensor, low-frequency oscillations (e.g., pilot-induced oscillations (PIO)) or vibrations (with a higher frequency) can be detected and fed into the SWHM model. The

⁶<http://reasoning.cs.ucla.edu/samiam>

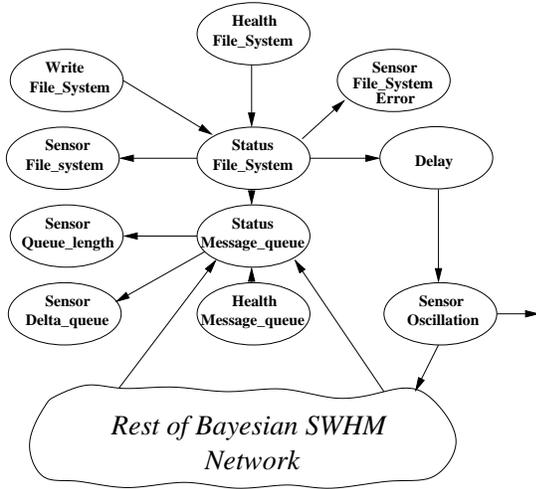


Figure 4. Partial Bayesian network for diagnosing faults potentially taking place in the software architecture shown in Figure 3.

SWHM reasoning then tries to perform a disambiguation on the cause of the oscillation or vibration.

This Bayesian network is compiled into an arithmetic circuit, which is integrated with the rest of the system as shown in Figure 2.

4.4 Results

Analysis of experimental runs with this architecture indicated that the system undergoing SHWM runs fine in the nominal case (Figure 5). However, the SWHM inference engine was instrumental in pointing toward the root cause of oscillations when pitch-up and pitch-down commands to the aircraft plant are affected by faults originating in the file system, causing the aircraft to oscillate up and down rather than maintain the desired altitude. For the purpose of our experiments, the file system was set to almost full at the start of the run, and as the system runs and controls are issued and logged, delays in executions start taking place at time $t = 30s$ (Figure 6) but no software errors are flagged. Eventually, altitude oscillations are detected by a fast Fourier transform performed on the altitude sensor readings shown in the middle panel of Figure 6. The bottom panel indicates that when the fast Fourier transform eventually detects oscillations around $t = 100s$, the SWHM infers that the posterior probability of good software health drops substantially, while the posteriors of good health of pitch and accelerometer systems are mostly high despite some transient lows. This indicates a low degree of belief in the good health of the software and that the most likely cause for a state with oscillations would be a software fault. For the purpose of this experiment, no additional pilot inputs were assumed.

SHWM can also be instrumental in disambiguating the root

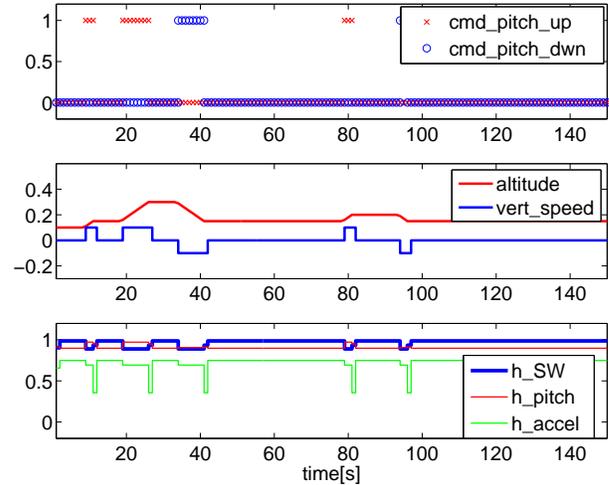


Figure 5. Temporal trace for the nominal case of file system based scenarios. The top panel shows pitch up and down commands to the aircraft. The middle panel shows the readings of altitude and vertical speed. The bottom panel shows the degree of belief in the good health of the accelerometer sensor (h_accel , green), of the pitch signal (h_pitch , red), and of the software (h_SW , thick blue line).

cause of oscillations when we add a pilot input node connected to the oscillation detection fast Fourier transform sensor node. The SWHM reasoner can then disambiguate the diagnosis by evaluating whether the fault is due to PIO or a software problem.

The SWHM models, which we have presented here are able to recognize and disambiguate known failure classes. In general, the handling of emergent behavior, i.e., the occurrence of events or failures that have not been considered or modeled, is an important task for a system-wide health management system. Such failures can occur if the system is operated in a new environment, or due to unexpected interaction between components.

Our SWHM approach can—albeit with some restrictions—detect and diagnose emergent behavior. If we model the software behavior using safety and performance requirements (in addition to specific pre-analyzed) failure modes, emergent behavior, which manifests itself adversely by violating safety requirements or lowers performance, can be detected and diagnosed.

In our experimental setting, relevant performance or safety requirements could be: no vibrations or oscillations should occur, and a smooth flight path without specific pilot input should not require substantial actuator activation. With the existing sensors and the reasoning capabilities of the Bayesian network, the failure scenario discussed above would raise an alarm due to the violation of these requirements.

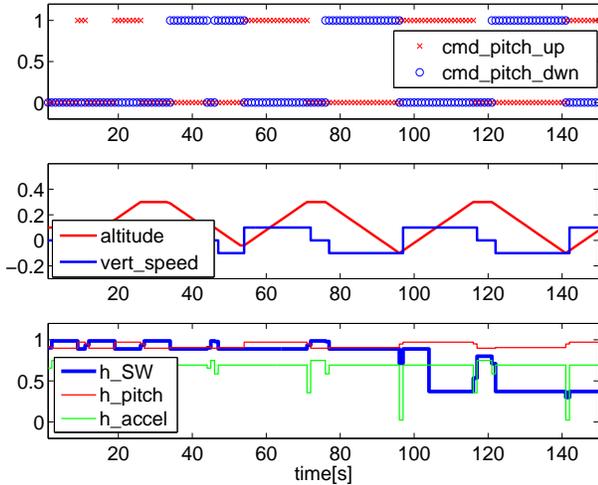


Figure 6. Temporal trace for a file system related fault scenario resulting in oscillations. The SWHM inference engine’s evaluation outputs show that the degree of belief in the good health of the system’s software (blue in bottom panel) substantially drops when oscillations are eventually detected by a fast Fourier transform at about $t = 100s$, after overflow of the file system resulted in delayed pitch up and pitch down command signals from the controller. Readings from the altitude sensor (blue in middle panel) show oscillating altitude starting at about $t = 30s$.

5. CONCLUSIONS

Software plays an important and increasing role in aircraft. Unfortunately, software (like hardware) can fail in spite of extensive verification and validation efforts. This obviously raises safety concerns.

In this paper, we discussed a software health management (SWHM) approach to tackle problems associated with software bugs and failures. The key idea is that an SWHM system can help to perform on-board fault detection and diagnosis on aircraft.

We have illustrated the SWHM concept using Bayesian networks, which can be used to model software as well as interfacing hardware sensors, and fuse information from different layers of the hardware-software stack. Bayesian network system health models, compiled to arithmetic circuits, are suitable for on-board execution in an embedded software environment.

Our Bayesian network-based SWHM approach is illustrated for a simplified aircraft guidance, navigation, and control (GN&C) system implemented using the OSEK embedded operating system. While OSEK is rather simple, it is We show, using scenarios with injected faults, that our approach is able to detect and diagnose non-trivial software faults.

In future work, we plan to investigate how the SWHM con-

cept can be extended to robustly handle unexpected and unmodeled failures, as well as how to more automatically generate SWHM Bayesian models based on information in artifacts including software engineering models, source code, as well as configuration and log files.

ACKNOWLEDGMENT

This work is supported by a NASA NRA grant NNX08AY50A “ISWHM: Tools and Techniques for Software and System Health Management”.

REFERENCES

- Adler, M. (2006). *The Planetary Society Blog: Spirit Sol 18 Anomaly*. Retrieved 02/2010, from <http://www.planetary.org/blog/article/00000702/>
- Chavira, M., & Darwiche, A. (2007). Compiling Bayesian Networks Using Variable Elimination. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)* (p. 2443-2449). Hyderabad, India.
- Darwiche, A. (2001). Recursive conditioning. *Artificial Intelligence*, 126(1-2), 5-41.
- Darwiche, A. (2003). A Differential Approach to Inference in Bayesian Networks. *Journal of the ACM*, 50(3), 280–305.
- Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge, UK: Cambridge University Press.
- Jensen, F. V., Lauritzen, S. L., & Olesen, K. G. (1990). Bayesian Updating in Causal Probabilistic Networks by Local Computations. *SIAM Journal on Computing*, 4, 269–282.
- Lauritzen, S., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems (with discussion). *Journal of the Royal Statistical Society series B*, 50(2), 157–224.
- Li, Z., & D’Ambrosio, B. (1994). Efficient Inference in Bayes Nets as a Combinatorial Optimization Problem. *International Journal of Approximate Reasoning*, 11(1), 55–81.
- Mengshoel, O. J. (2007). Designing Resource-Bounded Reasoners using Bayesian Networks: System Health Monitoring and Diagnosis. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)* (pp. 330–337). Nashville, TN.
- Mengshoel, O. J., Chavira, M., Cascio, K., Poll, S., Darwiche, A., & Uckun, S. (2010). Probabilistic Model-Based Diagnosis: An Electrical Power System Case Study. *IEEE Trans. on Systems, Man, and Cybernetics*, 40(5), 874–885.

- Musliner, D., Hendler, J., Agrawala, A. K., Durfee, E., Strosnider, J. K., & Paul, C. J. (1995, January). The Challenges of Real-Time AI. *IEEE Computer*, 28, 58–66. Available from citeseer.comp.nus.edu.sg/article/musliner95challenges.html
- Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.
- Poll, S., Patterson-Hine, A., Camisa, J., Garcia, D., Hall, D., Lee, C., et al. (2007). Advanced Diagnostics and Prognostics Testbed. In *Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07)* (pp. 178–185). Nashville, TN.
- Ricks, B. W., & Mengshoel, O. J. (2009). Methods for Probabilistic Fault Diagnosis: An Electrical Power System Case Study. In *Proc. of Annual Conference of the PHM Society, 2009 (PHM-09)*. San Diego, CA.
- Ricks, B. W., & Mengshoel, O. J. (2010). Diagnosing Intermittent and Persistent Faults using Static Bayesian Networks. In *Proc. of the 21st International Workshop on Principles of Diagnosis (DX-10)*. Portland, OR.
- Schumann, J., Mengshoel, O., & Mbaya, T. (2011). Integrated Software and Sensor Health Management for Small Spacecraft. In *Proc. SMC-IT*. IEEE.
- Shenoy, P. P. (1989). A valuation-based language for expert systems. *International Journal of Approximate Reasoning*, 5(3), 383–411.
- Zhang, N. L., & Poole, D. (1996). Exploiting Causal Independence in Bayesian Network Inference. *Journal of Artificial Intelligence Research*, 5, 301–328. Available from citeseer.nj.nec.com/article/zhang96exploiting.html



Johann Schumann (PhD 1991, Dr. habil 2000, Munich, Germany) is Chief Scientist for Computational sciences with SGT, Inc. and working in the Robust Software Engineering Group at NASA Ames. He is engaged in research on software health management, verification and validation of health management systems, and data analysis for air traffic control. Dr. Schumann's general research interests focus on the application of formal and statistical methods to improve design and reliability of advanced safety- and security-critical software. Dr. Schumann is author of a book on theorem proving in software engineering and has published more than 90 articles on automated deduction and its applications, automatic program synthesis, software health management, and neural network oriented topics.



Ole J. Mengshoel received the B.S. degree from the Norwegian Institute of Technology, Trondheim, Norway, in 1989, and the Ph.D. degree from the University of Illinois at Urbana-Champaign, Illinois, in 1999, both in computer science. He is currently a Senior Systems Scientist with Carnegie Mellon University (CMU), Silicon Valley, CA,

and affiliated with the Intelligent Systems Division, National Aeronautics and Space Administration (NASA) Ames Research Center, Moffett Field, CA. Prior to joining CMU, he was a Senior Scientist and Research Area Lead with US-RA/RIACS and a Research Scientist with the Decision Sciences Group, Rockwell Scientific, and Knowledge-Based Systems, SINTEF, Norway. His current research focuses on reasoning, machine learning, diagnosis, prognosis, and decision support under uncertainty - often using Bayesian networks and with aerospace applications of interest to NASA. He has published more than 50 papers and papers in journals, conference proceedings, and workshops. He is the holder of four U.S. patents. Dr. Mengshoel is a member of the Association for the Advancement of Artificial Intelligence, the Association for Computer Machinery, and IEEE.



Timmy Mbaya is a computer science student at the University of Massachusetts in Boston. His interests are Artificial Intelligence, probabilistic reasoning, and real-time embedded systems.