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

Johann Schumann[1], Timmy Mbaya[2], Ole Mengshoel[3]

[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 (BNs) to monitor the health of the on-board software and sensor system, and to perform advanced on-board diagnostic reasoning. We will focus on the approach to develop reliable and robust health models for the combined software and sensor systems.

1. INTRODUCTION

Modern aircraft more and more depend 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 lead to disastrous effects.

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 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 GN&C system did not only shut down that safety-critical component but also brought down communications, so the F-22's 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 as soon as they occur. In particular, a 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 interaction, status information about software components must be collected and processed as well.
- *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. During a recent incident (e.g., when one of the engines exploded on a Qantas A380), the pilot has to sort though literally hundreds of diagnostic messages in order to find out what happened. In addition, several diagnostic messages contradicted each other[2].

[1] http://www.af.mil/news/story.asp?storyID=123041567

[2] http://www.aerosocietychannel.com/aerospace

In this paper, we describe our approach to use Bayesian networks as the modeling and reasoning paradigm for SWHM. With a properly set-up 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 a SWHM system that is supposed to operate on-board the aircraft in an embedded environment must satisfy important properties: The implementation of the SWHM must have a small memory and computational footprint and must be certifiable. In this paper, we briefly will discuss issues for the verification and validation (V&V) of SWHM, which is an important prerequisite for any certification. Our approach using HM models, which have been compiled into arithmetic circuits are amenable to V&V. Finally, 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.

The reminder of the paper is structured as follows: Section 3. introduces Bayesian networks and how they can be for general diagnostics. In Section 3. demonstrate our approach toward 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 Bayesian health model to diagnose such scenarios, and present some 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 our SWHM), each random variable (or node) has a finite number of states and is parameterized by a conditional probability table (CPT).

-insight/2010/12/exclusive-qantas-qf32-flight-from-the-cockpit/

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 diagnosis for the system. Common BN queries of interest in this context include computing posterior probabilities and finding the most probable explanation (MPE). For example, an observation about an abnormal behavior of a software component could, by computing the MPE using a BN model of the system, 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 real-time performance. The compilation approach—which includes join tree propagation and arithmetic circuit evaluation—is attractive in resource-bounded systems.

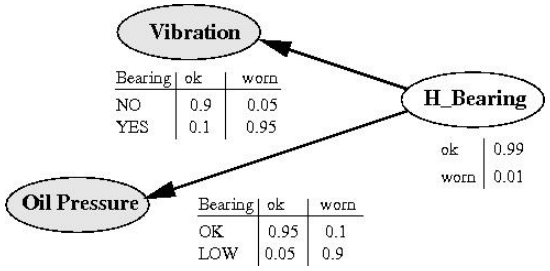

Figure 1. Simple Bayesian Network. CPT tables are shown near each node.

Let us consider a very simple example of a Bayesian network (Figure 1) as it could be used in diagnostics. Figure 1 shows the network and the CPT tables for each node. We have a node `H_Bearing` representing the health of a ball bearing in a diesel engine, a sensor node `Vibration` representing whether vibration is measured or not, and a node `Oil Pressure` 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 reflected in the two CPTs depicted next to the sensor nodes. For example, if there is vibration, this increases the probability that $p(\texttt{H\_Bearing} = worn)$. More formally, to obtain the health of the ball bearing, we input the states of the sensor nodes into the BN, and compute the posterior distribution (or belief) over `H_Bearing`. The prior distribution

of failure, as reflected in the CPT shown next to `H_Bearing`, is also taken into account in this calculation.

Using Darwiche's work on compiling Bayesian Networks into Arithmetic Circuits (Darwiche, 2009), the example network above would reflect a joint probability distribution as follows[3]. And for simplicity, let's replace all CPT entries with Θ_x (i.e., $\Theta_{ok} \leftrightarrow$ H_Bearing is ok, and $\Theta_{\sim ok} \leftrightarrow$ H_Bearing is worn). Let λ_i indicate whether evidence of a specific state is observed (i.e., $\lambda_v = 1$ means evidence of v (vibration) is observed, and $\lambda_v = 0$ means no evidence of v (no vibration) is observed). The probability distribution p**(H_Bear, Vib, Oil)** captured by the Bayesian network above is shown in Table 1.

H_Bear	**Vib**	**Oil**	p**(H_Bear,Vib,Oil)**
ok	v	op	$\lambda_{ok}\lambda_v\lambda_{op}\Theta_{v\|ok}\Theta_{ok}\Theta_{op\|ok}$
ok	v	∼op	$\lambda_{ok}\lambda_v\lambda_{\sim op}\Theta_{v\|ok}\Theta_{ok}\Theta_{\sim op\|ok}$
ok	∼v	∼op	$\lambda_{ok}\lambda_{\sim v}\lambda_{\sim op}\Theta_{\sim v\|ok}\Theta_{ok}\Theta_{\sim op\|ok}$
ok	∼v	op	$\lambda_{ok}\lambda_{\sim v}\lambda_{op}\Theta_{\sim v\|ok}\Theta_{ok}\Theta_{op\|ok}$
∼ok	v	op	$\lambda_{\sim ok}\lambda_v\lambda_{op}\Theta_{v\|\sim ok}\Theta_{\sim ok}\Theta_{op\|\sim ok}$
∼ok	∼v	op	$\lambda_{\sim ok}\lambda_{\sim v}\lambda_{op}\Theta_{\sim v\|\sim ok}\Theta_{\sim ok}\Theta_{op\|\sim ok}$
∼ok	v	∼op	$\lambda_{\sim ok}\lambda_v\lambda_{\sim op}\Theta_{v\|\sim ok}\Theta_{\sim ok}\Theta_{\sim op\|\sim ok}$
∼ok	∼v	∼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**(H_Bear, Vib, Oil)**

According to this joint probability distribution table, the first row ($\lambda_{ok}\lambda_v\lambda_{op}\Theta_{v|ok}\Theta_{ok}\Theta_{op|ok}$) representing the probability that the health of a ball bearing be okay (λ_{ok} = 1), and that vibrations and good oil pressure are observed (λ_v and λ_{op} = 1) would be 0.9% (given corresponding numerical CPT entries): $\Theta_{v|ok}\Theta_{ok}\Theta_{op|ok} = 0.1 * 0.99 * 0.95 = 0.09405$. Indicating a very low degree of belief in such a state. 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 health of a ball bearing be okay (λ_{ok} = 1), and that no vibrations and good oil pressure are observed ($\lambda_{\sim v}$ and λ_{op} = 1) is much higher (84%) as follows: $\Theta_{\sim v|ok}\Theta_{ok}\Theta_{op|ok} = 0.9 * 0.99 * 0.95 = 0.84645$.

Each of this network's individual posterior marginals is then given by:

$$\mathrm{p}(H\_Bear, Vib, Oil) = \prod_{\Theta_{s|x}} \Theta_{s|x} \prod_{\lambda_s} \lambda_s$$

where $\Theta_{s|x}$ indicates a state's conditional probability and λ_s indicates whether or not state s is observed.

Then summing all individual posterior marginals yields a multi-linear function—at the core of arithmetic circuit evaluation—referred to as the "network polynomial" f by Darwiche:

$$\begin{aligned} f = \; & \lambda_{ok}\lambda_v\lambda_{op}\Theta_{v|ok}\Theta_{ok}\Theta_{op|ok} + \\ & \lambda_{ok}\lambda_v\lambda_{\sim op}\Theta_{v|ok}\Theta_{ok}\Theta_{\sim op|ok} + \\ & \lambda_{ok}\lambda_{\sim v}\lambda_{\sim op}\Theta_{\sim v|ok}\Theta_{ok}\Theta_{\sim op|ok} + \\ & \lambda_{ok}\lambda_{\sim v}\lambda_{op}\Theta_{\sim v|ok}\Theta_{ok}\Theta_{op|ok} + \\ & \lambda_{\sim ok}\lambda_v\lambda_{op}\Theta_{v|\sim ok}\Theta_{\sim ok}\Theta_{op|\sim ok} + \\ & \lambda_{\sim ok}\lambda_{\sim v}\lambda_{op}\Theta_{\sim v|\sim ok}\Theta_{\sim ok}\Theta_{op|\sim ok} + \\ & \lambda_{\sim ok}\lambda_v\lambda_{\sim op}\Theta_{v|\sim ok}\Theta_{\sim ok}\Theta_{\sim op|\sim ok} + \\ & \lambda_{\sim ok}\lambda_{\sim v}\lambda_{\sim op}\Theta_{\sim v|\sim ok}\Theta_{\sim ok}\Theta_{\sim op|\sim ok} \end{aligned}$$

Or

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

where E indicates evidence of a network instantiation, $\Theta_{s|x}$ indicates a state's conditional probability within E, and λ_s indicates whether or not state s is observed within E.

Queries are then performed on the circuit using relevant algorithms. A bottom-up visitation of the circuit, from input to output, evaluates the probability of a particular evidence on the state of the network. And a top-down visitation of the circuit, from output to input, differentiates the circuit output for every input, and can also provide information about how change in a specific node affects the whole network, which is sensitivity analysis.

3. BAYESIAN NETWORKS FOR SOFTWARE HEALTH MANAGEMENT

At a first glance, the SWHM does look very similar to a common integrated vehicle health management system (IVHM): sensor signals are interpreted to detect and identity any faults, which are being 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 principal differences between FDIR for hardware subsystems 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 effect the current software module (e.g., when a division-by-zero is detected), there are situations where the effects of software fault manifest themselves in an entirely different subsystem, as discussed in the F-22 example above. Due to this reason and the fact that many software problems occur due to problematic SW/HW interaction, 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 set up 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 nodes 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 (or subcomponents)

[3] In the following, we abbreviate for the bearing: worn = ∼ok, for the Oil Pressure: OK = op, and LOW = ∼op, and vibration by v.

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 preprocessing 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.

In the following section, we first will discuss the structure of our Bayesian health models before we discuss sources of (software) sensor data and preprocessing.

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 distinct states.

CMD node C the nodes comprise the "commanded input" to the network. Signals sent to these nodes are handled as ground truth and are used to indicate modes, actions, or (known) states. For example, a node `WRITE_TO_FS` notifies that 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[4]. 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 is an input node similar to the CMD node. However, 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, sensor nodes are typically connected with a health node, describing the health of a signal node.

HEALTH node H The health nodes are nodes, which reflect the health status of a sensor or component. Its posterior probabilities comprise the output of the health model. A health node can be binary (`OK, BAD`), or can have more states that reflect the health in more detail. 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.

[4]If 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.

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.

Sensor	H	$p =$ for `FS_status`			
		`E`	`OK`	`full95`	`full`
`empty`	`OK`	0.9	0.05	0.01	0.01
`OK`	`OK`	0.1	0.6	0.2	0.1
`almost_full`	`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
`almost_full`	`bad`	0.5	0.5	0	0
`full`	`bad`	0.5	0.5	0	0

Table 2. Sensor node for `Filesystem_capacity` with attached health node H and status node `FS_status`.

3.1.2 Arrows

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.

For example, if 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. After all, we use a sensor to better understand what is happening in a system. 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 modelling of state (as is done in $\{H, U\} \rightarrow S$ 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 such a case. Because the sensor node (for `Filesystem_capacity` has two parents (a status node `FS_status` and a health node (with states `OK, bad`)), 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 columns refer to the states of the `FS_status` node. In this particular example, a bad file system sensor does not recognize that the file system might become full.

3.2 Software Sensors

Information that is needed to reason about software health must be extracted from the software itself and all components, which interact with the software, i.e., hardware sensors, actuators, the operating system, middle ware, and the computer hardware. Different software sensors provide information about the software on a different level of granularity and abstraction. Table 3 gives an impression on 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 run havoc and consumes too many resources (e.g., CPU time, inter process communication, etc.), which in turn can lead to failures related to the control task. We therefore extract a multitude of different information about the software and its behavior as shown in Table 3.

Software	
Errors	flagged errors and exceptions
Memsize	used memory
Quality	signal quality
Reset	Filter reset (Naviation)
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 processes
M_free	available memory
D_free	percentage of free disk space
IPC	amount of available IPC
Semaphores	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 goal of preprocessing is to extract important information from the (large amount of) sensor data. Most of the preprocessing functions aim just toward data and dimensional reduction, and to convert the actual software sensor values into observed states of the health model. The latter is necessary since all SWHM nodes have a discrete state. For example, the sensor for the file system has the states `empty, OK, almost_full, full`. Preprocessing steps, which extract temporal features from the 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 with Bayesian networks (which are complex in design and execution), we are able to use much simpler static health models and handle all temporal aspects during preprocessing.

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

discretization A continuous value is discretized using a number of monotonically increasing thresholds. For the file system sensor, an example is shown in Table 4.

min/max/average The minimal or maximal value, or the mean, is taken.

moving min/max/average A moving min/max/mean value (with a selectable window size) is taken.

sum 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 Advanced time series analysis can be used as a preprocessing step for SWHM. For example, 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)	Status
$0 \leq df < 5\%$	`empty`
$5 \leq df < 80\%$	`OK`
$80 \leq df < 99\%$	`almost_full`
$99 \leq df$	`full`

Table 4. Discretization with thresholds

4. Demonstration Example

4.1 System architecture

For demonstration purposes, we have implemented a simple system architectures on a platform that reflects real-time embedded execution typical of aircraft and satellite systems. Trampoline[5], an emulator for the OSEK[6] real-time operating system (RTOS)—widely used within industry for embedded control systems—is used as a platform rather than other RTOS specifically established in the aerospace industry such as Wind River's VxWorks and GreenHills' INTEGRITY because its capabilities and easy availability was 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/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/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 real-time requirements (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.

4.2 Example Scenario

An experimental scenario architecture to study file system related faults such as the Mars rover Spirit reboot cycle incident (Adler, 2006) has been implemented on this basic platform. A short time after landing, the Mars rover SPIRIT encountered repeated reboots, because a fault during the booting process caused a reboot again. According to reports (Adler, 2006) an on-board file system for intermediate data storage cause 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 solved successfully.

[5] urlhttp://trampoline.rts-software.org/

[6] urlhttp://www.osek-vdx.org/

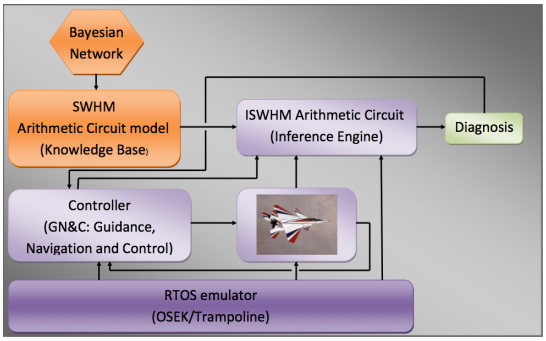

Figure 2. Demonstration System Architecture. The Bayesian Network model is compiled (before deployment) into an arithmetic circuit representing the knowledge base. The real-time operating system schedules three tasks: the controller, the plant, and the SWHM inference engine

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, which 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, a flawed software architecture with a global message queue that buffers all controller signals and logs them in the file system (blocking) before sending them is designed (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: guidance, controller, message queue, file system, and plant. On-board camera and transmitter are shown but not used in the experiments described in this paper.

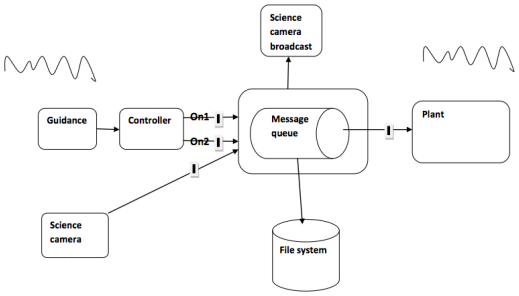

Figure 3. Software Architecture for file system related fault scenarios.

Here, we are running the following scenario: The file system is set to almost full. Subsequent control messages, which are being logged, might stay longer in the message queue, potentially causing overflow of the message queue or dropping of messages. However, a simple 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 affected in a non-obvious way.

Other possible scenarios with this setup are:

- The pilot, 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 by the SWHM task.
- 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/missed signals/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 out 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 is designed (Figure 4) using the SamIam tool[7]. A modular Bayesian network design approach is 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. Other subnetworks are then added to this underlying 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 causing oscillations of an aircraft or satellite are: the Write_File_System command node; the Health_File_System health node; the Status_File_System status node; the Sensor_File_System sensor node; the Sensor_File_System_Error sensor node; the Status_message_Queue status node; the Sensor_queue_length sensor node; the Sensor_delta_queue sensor node; the Health_message_Queue health node; the delay status node; and the Oscillation sensor node.
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 reflect the their unobservable states while their sensor nodes indicate sensor readings as to their states.

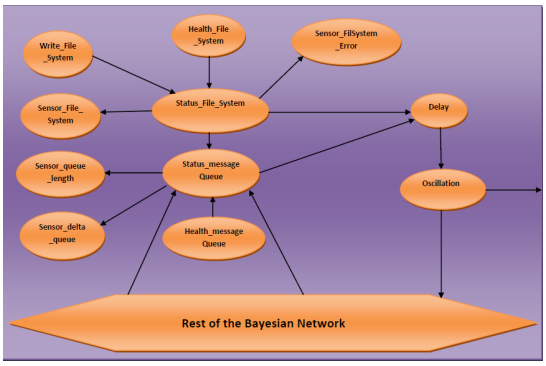

Figure 4. Partial Bayesian network for file system related architecture.

The only non-standard software sensor nodes in this SWHM model are the delay node and the sensor for oscillations detected by a Fast Fourier Transform. The delay node is an unobservable status node whose degrees of belief in delayed signals from the file system and the message queue given their status factor into evaluation of posterior marginals to determine the root causes of plant oscillations. The Fast_Fourier node is a sensor node whose input is from a fast Fourier transform software module performing time-series analysis to detect cyclic events such as oscillations in aircraft altitude when the aircraft pitch command signals are delayed. These two additional nodes are instrumental in inference to determine the most likely cause of a plant oscillations. Given that PIO (Pilot Induced Oscillations) can also be the source of plant oscillations, we can add yet another node to this network and connect it to the fast Fourier sensor node to factor pilot input in posterior marginals evaluations in order to disambiguate the cause of plant oscillations.

This Bayesian network is compiled into an arithmetic circuit whose definition serves as the SWHM model (the knowledge base), and is integrated with the rest of the system (tasks including controller, plant, and the Inference engine) running over the RTOS. The Bayesian network model is compiled "offline"—only once—into an Arithmetic Circuit.

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 maintaining the

[7] http://reasoning.cs.ucla.edu/samiam

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=30 (Figure 6). 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 = 100, the SWHM infers that the posterior probability that the health of the software is good is low as it substantially drops while the health of pitch and accelerometer systems are mostly high despite some transient lows. This indicates a low degree of belief in the 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.

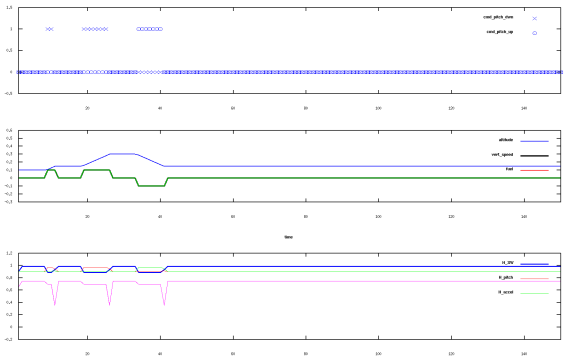

Figure 5. Temporal trace for the nominal case of file system based scenarios. The degree of belief in the health of the system software, in blue, remains high (bottom panel)

SHWM can also be instrumental in disambiguating the root 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 Pilot Induced Oscillations (PIO), or rather some software failures.

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 discuss 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

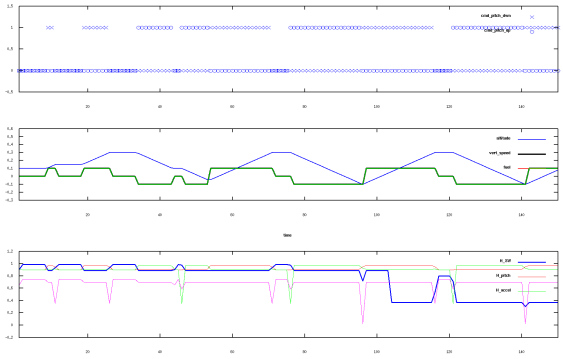

Figure 6. Temporal trace for a file system related fault scenario resulting in oscillations. The SWHM inference engine's evaluation outputs show that the degree of belief in the health of the system's software (blue in bottom panel) substantially drops when oscillations are eventually detected by a fast Fourier transform at about t=100, 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=30.

layers of the hardware-software stack. Bayesian network system health models, compiled to arithmetic circuits, are suitable for on-line execution in embedded software systems.

Our Bayesian network-based SWHM approach is illustrated for a simplified aircraft guidance, navigation, and control (GN&C) system implemented using the OSEK[8] embedded operating system. While OSEK is rather simple, it is extensively applied in the automotive and industrial sectors. We show, using scenarios with injected faults, that our approach is able to detect and diagnose software faults.

In future work, we plan to investigate how the SWHM concept 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.*

Chavira, M., & Darwiche, A. (2007). Compiling Bayesian Networks Using Variable Elimination. In *Proceedings*

[8]Specifically, we are using the Trampoline implementation of OSEK—see `http://trampoline.rts-software.org`.

of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07) (p. 2443-2449). Hyderabad, India.

Darwiche, A. (2001). Recursive conditioning. *AI*, *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. , *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. , *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.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann.

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.

Shenoy, P. P. (1989). A valuation-based language for expert systems. *AR*, *5*(3), 383–411.

Zhang, N. L., & Poole, D. (1996). Exploiting Causal Independence in Bayesian Network Inference. *JAIR*, *5*, 301-328.