Integrated Software and Sensor Health Management for Small Spacecraft

Johann Schumann
SGT, Inc./ NASA Ames
Moffett Field, CA, USA
Email: Johann.M.Schumann@nasa.gov

Ole J. Mengshoel
Carnegie Mellon University
Moffett Field, CA, USA
Email: ole.mengshoel@sv.cmu.edu

Timmy Mbaya
U. of Massachusetts
Boston, MA, USA
Email: timstim@mail.com

***Abstract*—Despite their size, small spacecraft have highly complex architectures with many sensors and computer-controlled actuators. At the same time, size, weight, and budget constraints often dictate that small spacecraft are designed as single-string systems, which means that there are no or few redundant systems. Thus, all components, including software, must operate as reliably. Faults, if present, must be detected as early as possible to enable (usually limited) forms of mitigation. Telemetry bandwidth for such spacecraft is usually very limited. Therefore, fault detection and diagnosis must be performed on-board. Further restrictions include low computational power and small memory.**

In this paper, we discuss the use of Bayesian networks (BNs) to monitor the health of on-board software and sensor systems, and to perform advanced on-board diagnostic reasoning. Advanced compilation techniques are used to obtain a compact SSHM (Software and Sensor Health Management) system with a powerful reasoning engine, which can run in an embedded software environment and is amenable to V&V. We successfully demonstrate our approach using an OSEK-compliant operating system kernel, and discuss in detail several nominal and fault scenarios for a small satellite simulation with a simple bang-bang controller.

***Index Terms*—Health Management, fault detection and diagnosis, Bayesian network**

I. Introduction

The impact of small satellites on science is growing at a tremendous pace. Rather than a few large and costly satellites, many small and low-budget satellites are being used to answer specific science questions.

The use of small satellites has impact on the development process as well. High modularization enables the developers of small satellites to obtain components from different vendors for optimal cost and performance. Systems integration and software development must proceed rapidly to meet project budget and schedule. However, even very small satellites consist of a number of complex subsystems (propulsion, avionics, communications, and science instruments), which must work together flawlessly. Budget constraints usually mandate "single string" design. This means that there are no redundant hardware or software components, which allow the system to detect failures (e.g., by a voting scheme) and to switch to a working component in the case of faults. Software to detect, isolate, and mitigate failures are highly primitive or missing altogether in current small satellite projects.

Clearly, a powerful FDIR (Fault Detection, Isolation, Recovery) or ISHM (Integrated System Health Management) system for hardware and software has great potential for ensuring the operational reliability of such satellites.[1] The role of software in modern satellites is increasing, just as it is for other spacecraft, aircraft, cars and other machinery; more and more functionality of the system is realized by using software rather than by dedicated (typically heavy or expensive) hardware components. The amount and complexity of embedded software running on satellites has increased tremendously over the last few years. With budget limitations for development as well as for verification and validation (V&V), problems are almost guaranteed. In this paper, we focus on small spacecraft system health management, including both software and sensor health management.

A number of missions have failed due to software errors or problematic software-hardware interactions. Although SSHM is not able to detect all such problems, the reliability of a software system can be enhanced with a system that can detect and diagnose software failures. The following examples illustrate how simple software problems can jeopardize expensive missions.

In 1999, the Mars Polar Lander (MPL) crashed onto the surface of Mars. An incident investigation concluded that "the most probable cause of the failure was the generation of spurious signals when the lander legs were deployed during descent. The spurious signals gave a false indication that the spacecraft had landed, resulting in a premature shutdown of the engines[...]." [1]. It seems that the software was ill-equipped to deal with such spurious signals. An on-board SSHM could have taken additional available information into account (e.g., from the radar altimeter) and potentially have mitigated the engine shutdown and consequently the crash.

A short time after landing, the Mars rover SPIRIT encountered repeated reboots, because a software fault during the booting process caused an immediate reboot. According to reports [2], 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 solved

[1] Recovery or mitigation aspects, which are especially important for fully autonomous operations, are beyond the scope of this paper.

successfully. This example shows how, despite careful V&V, hard to detect errors can still remain dormant in the software. This example also shows that certain kinds of software failures could be detected by the SSHM system by monitoring and reasoning before the actual faults occur.

A powerful, on-board Software and Sensor Health Management (SSHM) has the potential to detect many possible faults as soon as they occur and can thus substantially contribute to overall mission safety and reliability. Such a SSHM

- *monitors the behavior of the software, the operating system, and the attached sensors during system operation.* Information about operational status, signal quality, quality of computation, reported errors, etc., is collected and processed on-board. Most of this information is readily available without the need to instrument or otherwise modify the software.
- *performs substantial diagnostic reasoning in order to identify the most likely root cause(s) for the fault and a quality measure for that result.* This diagnostic capability is extremely important, because telemetry bandwidth of a small satellite is very limited. So, only a small subset of the system information can be down-linked, which makes accurate ground-based failure diagnosis impossible.

Simple diagnostic routines, as often used with current systems, typically process symptoms in isolation, which can result in incorrect diagnoses or a large number of contradictory results. This problem became evident in a recent A380 incident[2], in which the pilots reported more than 400 diagnostic messages, some of which contradicted each other.

Finally, a proper probabilistic treatment of the diagnosis process, as we accomplish with our Bayesian approach [3], [4], can not only merge information from multiple sources but also provide a posterior distribution for the diagnosis. We note that this approach has been very successful for electrical power system diagnosis [5]–[7].

Such a SSHM system must, to be useful for a small satellite, satisfy several important properties. The SSHM must:

- *have a small memory and resource footprint.* Here, resources do not only mean low computational requirements, but the SSHM also needs to "compress" the system and software status into few, relevant, and reliable pieces of health information, which can be transmitted using a low telemetry bandwidth. Our SSHM system thus performs an intelligent compression of raw sensor data into system health information.
- *be able to fuse information from hardware sensors and software monitors.* For an SSHM system, it is not sufficient to just monitor software and hardware separately, because many software problems occur due to problematic software-hardware interactions. For example, the Ariane 5 (software) error [8] was triggered because the hardware sensor produced sensor readings with a range of values that was larger than acceptable by the SW, which had been constructed for the much smaller Ariane 4. In our SSHM approach, we use Bayesian networks to model SW and HW, as well as their interaction, in a probabilistically principled way.
- *exhibit a low number of false positives and false negatives.* False alarms (false positives) can produce nuisance signals; missed adverse events (false negatives) can endanger mission success. We ensure a high-quality SSHM by taking a BN-based approach, where there is a clear mapping from the structure of the satellite hardware and software to the structure of the BN.
- *should not require instrumentation or other changes to the software.* Most of the information required by the SSHM is already available within the software system and can be read from global storage or buses. Thus a powerful SSHM can be developed, which does not interfere with other software. Where necessary and suitable, specific software sensors within the software can provide additional information to the SSHM.
- *be V&V-able.* Any software that is running on-board a satellite must undergo rigorous V&V. Our approach of using SSHM models, in the form of BNs that have been compiled into arithmetic circuits, is amenable to V&V.
- *constructed in a modular, hierarchical, and re-usable way.* In order to support modular construction of a small satellite, the health management modeling paradigm also must support modular, hierarchical, and reusable models. Bayesian networks (BNs) can provide these features, but research questions in this area need to be addressed (see Section VI).

Once a fault has been detected and its causes diagnosed properly, a multitude of different autonomous or ground based mitigation strategies could be applied to deal with that problem. However, successful mitigation requires a reliable detection and diagnosis of the software faults. Thus, in this paper, we focus on detection and diagnosis part of SSHM.

The paper is structured as follows: Section II discusses Bayesian networks and their compilation into arithmetic circuits. In Section III we demonstrate our approach, and discuss the system architecture, the Bayesian network SSHM model, and experiments. In Section IV we address V&V issues, both at the model level (Bayesian network) and code level (arithmetic circuit). In Section VI we conclude and identify future work.

II. Software and Sensor Health Management

At a first glance, an SSHM looks very similar to a traditional integrated vehicle health management system (IVHM) that monitors vehicle hardware: sensor signals are interpreted to detect and identify faults, which are reported. Such FDIR systems are nowadays commonplace in the aircraft and automotive industries. It seems like it would be straight-forward to attach a software to be monitored (host software) to such a FDIR. However, there are several important differences: usually, software faults do not develop gradually over time (e.g., like an oil leak); instead, they occur instantaneously.

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

Furthermore, many faults occur due to problematic SW/HW interaction. Thus, both software and hardware must be monitored, in an integrated fashion.

Based upon necessary features and requirements as laid out in Section I, we are using Bayesian networks to set up software and (hardware) sensor health management models.

A. Bayesian Networks

We take a model-based approach to SSHM, and assume for the purpose of this paper that Bayesian networks (BNs) [3], [4] are being used. A BN represents a multi-variate probability distribution by organizing random variables in a directed acyclic graph—an approach that enables efficient computation as long as the graph is relatively sparse. The type of computation we are interested in is fault detection and diagnosis. By observing time series for the commands and sensors of a small spacecraft (jet control, IMU, GPS, voltage, current, temperature, and software sensors), it can be relatively easy to diagnose sensor or software failures. On the other hand, fault diagnosis can also be non-trivial when there are multiple faults and relatively few sensors available. The BN-based approach we discuss in this paper handles such non-trivial settings.

A typical SSHM BN, like the one used in the present case study, contains command nodes, sensor nodes, and health nodes [7], [9]. After updating with evidence, the posterior distribution over the BN's health nodes corresponds to the best system health estimate the BN is providing. Commands and sensor values may be continuous (e.g., voltage, temperature) or discrete. In order to use continuous commands and sensor values in a Bayesian network restricted to use only discrete random variables, the continuous values must be discretized into values that correspond to the states of the random variables in the BN.

There are several benefits to using Bayesian networks. First, they can be auto-generated, manually constructed, or machine learned [3], [4]. Also, they easily support the detection of multiple faults, can be compiled into arithmetic circuits for exact computation in an embedded environment; and a broad range of efficient algorithms have been developed.

We note that traditional diagnostics ("flight rules") can easily be modeled as a Bayesian network. Here, CPT values of the nodes are restricted 0 or 1. With our approach, we are thus able to incorporate flight rules into our models and reason about them. For example, different flight rules (or even parts thereof) can be "weighted" differently and their probability of occurring merged in a principled way. In addition, with an appropriate SSHM model, BNs can be exploited for sensor validation as well as for diagnosis [10].

B. Compilation and SSHM architecture

Bayesian networks are often compiled, off-line, to secondary data structures such as clique trees or arithmetic circuits [4]. These secondary data structures are then used for on-line computation. For resource-restricted and embedded settings, this compilation approach is especially suitable

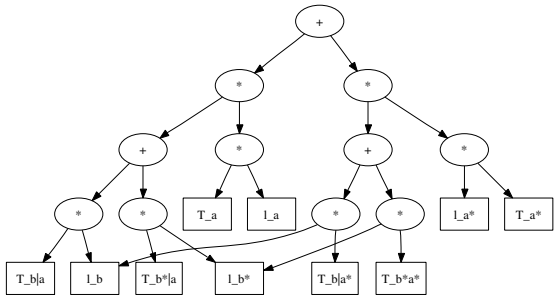

Fig. 1. Arithmetic circuit as constructed from a Bayesian network.

[11]. During on-line computation, the arithmetic circuit-based diagnostic software reads commands and sensor data, performs preprocessing including discretization, inputs evidence into the arithmetic circuit, evaluates it, and outputs diagnostic results based on the posterior distribution over the health random variables [5]–[7].

As an example, we mention the very successful use of arithmetic circuits, compiled from BNs, for fault diagnosis in electrical power systems that are representative of those found in aerospace vehicles [5], [6]. While, in general, there can be significant growth in the size of clique trees and arithmetic circuits as BN connectivity increases [12], we have found this not to be a problem in electrical power systems [7], [13]. Typical health models for this domain contain around 500 nodes with between 2 and 16 discrete states (with a mean of 2.85 states/node). The arithmetic circuit (including code for the execution) is around 600kBytes in size and typical queries are evaluated in about $500\mu s$ (on a 2.3GHz MacBook Pro). The calculation requires floating point additions and multiplications; the use of fixed-point arithmetic might cut down processing time substantially.

III. System Health Management Demonstration

A. System Architecture

We will demonstrate our SSHM with a small satellite example. The satellite is point-mass and has only two cold gas jets pointing forward and aft. These jets can be used to slow down or accelerate the spacecraft. A simple bang-bang controller with dead-band is used to keep it at a given velocity. As hardware sensors, we have an accelerometer and a second sensor capable of providing estimates for position and velocity ($\vec{x}$,$\vec{v}$), e.g., a star tracker or a GPS system. For simplicity of this example, the spacecraft is assumed to move along a one-dimensional trajectory.

For spacecraft control, we use a simplified software architecture depicted in Figure 3. An OSEK-compliant operating system kernel is used for all operations. Albeit very primitive, operating systems that are compliant with the OSEK standard[3] are widely used in the automotive industry. The individual components of our GN&C system are implemented as tasks, running at a fixed schedule. Communication between tasks is

[3]http://www.osek-vdx.org

Name	Type	States
Cmd_f_jet	CMD	idle, on
Cmd_r_jet	CMD	idle, on
Sensor_f_flow	SENSOR	zero, small, large
Sensor_r_flow	SENSOR	zero, small, large
Sensor_fuel	SENSOR	ok, low, empty
Sensor_acc	SENSOR	negative, zero, positive
Health_f_jet	HEALTH	ok, low_eff, bad
Health_r_jet	HEALTH	ok, low_eff, bad
Health_SW	HEALTH	ok, ctrl_fault

TABLE I
SENSOR SIGNALS FOR SSHM (SELECTION)

performed on a low level using global variables guarded by OSEK resources. For the sake of demonstration, the control signals are transferred using message passing with fixed-length message queues.

We have chosen this set-up specifically for demonstration purposes, and note that our SSHM approach is totally independent of a specific software or hardware architecture. An SSHM system can be run on small embedded systems (as demonstrated here), on larger RTOS systems (e.g., using ARINC 653 [14]) or even on Unix-style systems.

B. The SSHM and the System Health Model

The SSHM itself is implemented as a separate task (low priority, low frequency) and has access to operational data as collected by hardware and software sensors. In our case, all data are available in global variables and the proper GN&C software did not have to undergo any SSHM-specific modifications or instrumentation. The assembled data are preprocessed and discretized as necessary and fed into the evaluator for the SSHM arithmetic circuit. The actual computation of health status, which is performed at regular intervals, is guaranteed to be time-bounded. The small SSHM models as employed in our demonstration example use around 10kBytes of memory and the evaluation takes around $7\mu s$ (2.3GHz CPU). After evaluation of the arithmetic circuit, the resulting posterior distributions on health nodes can then be fed into the telemetry stream or used to perform certain recovery actions. In this paper, we do not discuss possible recovery strategies.

Table I gives a selection of software and hardware sensor signals and health nodes, which are used for our demonstration examples. In each case, we give its origin (hardware sensor, software sensor, health nodes, or command signal) and its discrete states.

Note that, in general, additional preprocessing steps might take place to derive the appropriate data from the stream of sensor signals (e.g., moving average, minimum, maximum).

Figure 2 depicts an excerpt of a Bayesian network model for a small spacecraft (as shown by the tool SamIam[4]). There are two jets, a fuel tank, two fuel sensors, and an accelerometer. The operation of the two jets is similar, so we just describe how one jet works. A jet is controlled through a command (idle

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

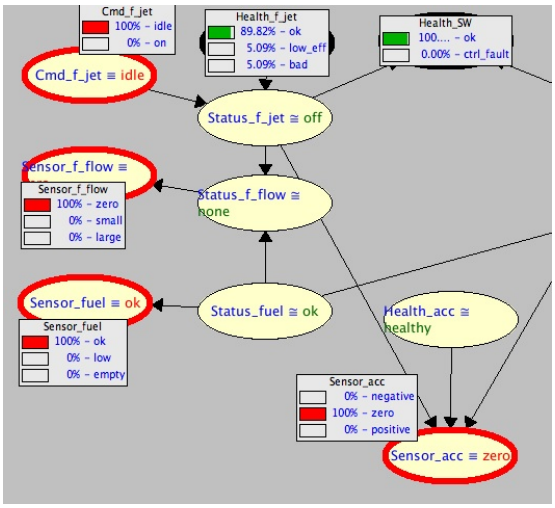

Fig. 2. Excerpt from the SSHM model for the experiment described in this paper. Nodes are overlaid with boxes that indicate the (posterior) probability of the state of each node. Nodes with a red border are sensor or command (input) nodes, those hidden behind the table are health (output) nodes. The status nodes represent the unobservable status of sensors, components and the software.

or on) that controls the flow of fuel from the fuel tank into the jet. The firing of the jet depends on the fuel flow. There is an accelerometer that senses the net acceleration (from both jets). Sensor nodes for the fuel (level and flow) are connected to the network. Since none of the sensors can be guaranteed to work properly, health nodes (e.g., Health_acc for the accelerometer sensor) are used to model possible faults of the sensors.

Modeling of software health requires taking into account information from various sources: hardware sensors, software sensors, and operating system (OS) sensors. Hardware sensors usually provide signals and information about their status (e.g., OK, noisy, bad). Software sensors monitor the behavior of a single software component or software signal. Typical status information includes triggered exceptions, numerical problems, failed assertions, etc. Specific software "intent", e.g., starting to write to a file, allocating dynamic memory, is useful to detect certain kinds of software errors. For example, a situation where the software intends to write to a file, but the file system does not change and is not full might indicate a problem in opening the file or a blocked write.

In addition to these discrete software signals, continuous information about the software component is collected, indicating the quality of the signals and computations. Typical examples include residual errors and quality information as produced by a Kalman filter (the diagonal of the process covariance matrix).

Finally, the status of the operating system can provide useful information for detecting and identifying software faults. Whereas a simple OS, like OSEK, only provides very little status information, more complicated OSes yield a wealth of important information. For SSHM, we might monitor variables (and their changes) such as CPU-load, free memory, global resources, length of message queues, stack size, number of

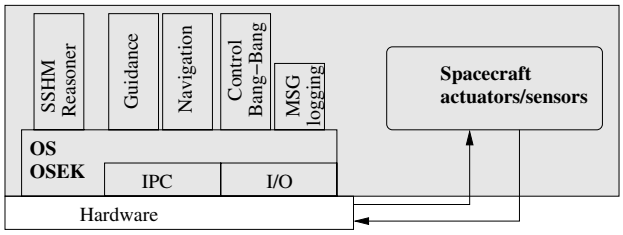

Fig. 3. Demonstration architecture: individual components of the GN&C system and the SSHM are implemented as OSEK tasks; communication between tasks is performed using global variables (resource-protected) or message queues. For simulation purposes, the plant model for the spacecraft was also implemented as a task, which runs with a 100ms cycle time. GN&C tasks are executed every 500ms, the SSHM every 200ms. The entire compiled software (grey box) was executed on a Mac OSX system.

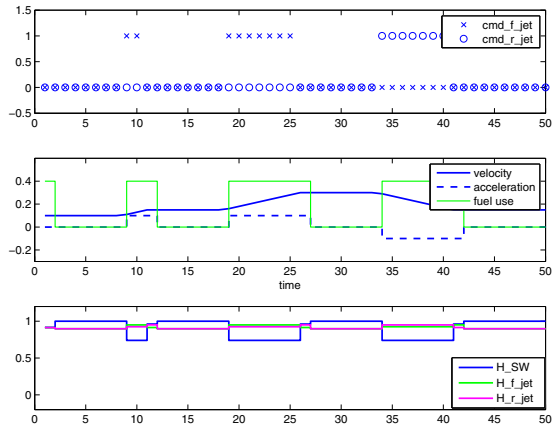

Fig. 4. Nominal trace of spacecraft status. Different speeds were commanded at different times. The bang-bang controller produces jet commands (1=on) for the forward (x) and reverse (o) jet (top). Measured sensor signals: acceleration (dashed), fuel consumption signal (0.4=jet(s) on) as well as true (unobservable) spacecraft velocity are shown in the middle panel. Bottom panel shows posterior probabilities for the health nodes for the software (H_SW), for the forward (H_f_jet), and the reverse (H_f_jet) jet. All probabilities are close to 1 indicating a healthy system state.

missed events, status of file system, process creation and deletion, among others. Albeit this information only provides a global situation snapshot, our Bayesian health models can use these data to disambiguate potential failure causes using a principled statistical approach.

What is the learning curve (or effort) associated with building BNs for system health management? Many tools and techniques are available to support BN model construction. More than a dozen software tools, both commercial and academic, exist, so this is a well-established technology. [5] Considering BNs for SSHM more specifically, more targeted tools and techniques also exist [5]–[7], [9]. For example, we have developed a compositional approach where BNs are auto-generated from a high-level specification [7], [9]. The main benefits of this auto-generation approach are that the SSHM developer no longer needs to understand BNs, just a much simpler domain-specific language, and the development effort is significantly reduced.

C. Experiments

The compiled SSHM model and the evaluator can be compiled and linked with the other software components and the OSEK kernel. When we execute this system in simulation, an additional task logs important internal signals, so we can track the entire behavior of the spacecraft (plant), the GN&C software, and the SSHM. Figure 4 shows a typical temporal trace of the system: commands to change velocity (top panel) result in firings of jets. The measured velocity, acceleration, and fuel use are shown in the middle panel. The bottom panel shows the posterior probability of the SSHM health nodes. Values close to 1 correspond to healthy software and components.

In Figure 5, we look at two different failure traces. Here, initially the same change in velocity as in Figure 4 is commanded: commanded firing at around $T = 8$ and $T = 18$ causes measured acceleration and thus an increase in velocity.

The left trace is a situation where we measure no acceleration, because a no change-velocity command had been given. Initially, health nodes for the jets, the accelerometer, and the software all show high-probability values for "healthy". However, after $T = 23$, no acceleration can be measured, despite the fact that the jet is still supposed to fire (Cmd_f_jet is set). Two things could have caused this: the accelerometer could have produced bad measurements, the jet might not be working, or (the unmodeled situation that) the spacecraft has landed and jet firing does not produce any acceleration. The Bayesian network can easily disambiguate these scenarios and comes up with the (injected) most probable fault—an nonoperational (e.g., iced-up) jet—because it detects that no fuel has been consumed. The health node for the jet (H_f_jet) clearly indicates this situation.

In Figure 5 (right), we again start out with the original healthy scenario. However, around $T = 42$, no acceleration is measured. However, fuel is consumed, and both thrusters should fire (Cmd_f_jet and Cmd_r_jet are set). This situation indicates that the thrusters are operational, but their actions cancel out. As the health model reflects the fact that at no time both thrusters should be operational, reasoning reveals that there is a software problem. In fact, the injected failure is a mess-up of the thresholds of the bang-bang controller.

The scenarios discussed above only touch an illustrative subset of the overall SSHM model.

IV. Toward V&V of SSHM

All software running on a spacecraft must undergo rigorous verification and validation (V&V) as well as a flight readiness review. Although no formal certification (e.g., according to DO-178B for manned aircraft) is necessary, there is a need to make sure that an SSHM system never, under any circumstances, causes problems for the overall system. In particular, it must not corrupt any data in memory, cause timing overruns, or crash the operating system. In addition, the

[5]For an overview see http://www.cs.ubc.ca/~murphyk/Bayes/bnsoft.html.

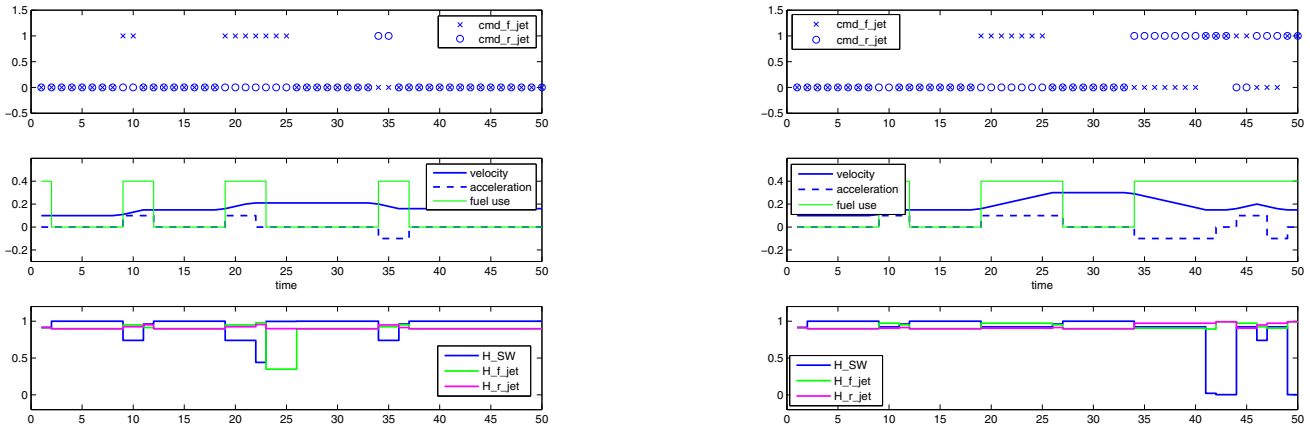

Fig. 5. System and SSHM traces for two scenarios with injected faults. Left: the forward jet was rendered nonoperational at T=23. At the same time, the signal H_f_jet falls below 0.5, which indicates a clear problem with the forward jet. Right: the seeded software error of overlapping lower and upper limits of the bang-bang controller could be detected in the deceleration phase. The H_SW signal goes gets very low, indicating a software problem. The top panel in this scenario shows that both thrusters are commanded on at the same time (after T=41).

SSHM should have a low rate of false alarms (false positives) and missed adverse events (false negatives). False alarms mean the SSHM is reporting failures despite a perfectly working system. Although such a situation is not necessarily dangerous (see, e.g., an incorrectly lit "Check Engine" light in a car), it could lead to unnecessary mission restrictions or to operators ignoring future alarms or warnings. Missed adverse events, on the other hand, could result in a mission failure, because undetected problems can potentially lead the spacecraft into an unrecoverable state.

For our model-based approach, we have to develop new approaches toward V&V, and perform analysis and V&V on the model level (Bayesian networks) as well as the code level (arithmetic circuits and evaluator). A sketch of a possible V&V process is shown in Figure 6. We think that this separate analysis can be justified because in most approaches, the SSHM model is translated or compiled into a highly compact and efficient data structure, which is then accessed by the SSHM engine to compute system health state.

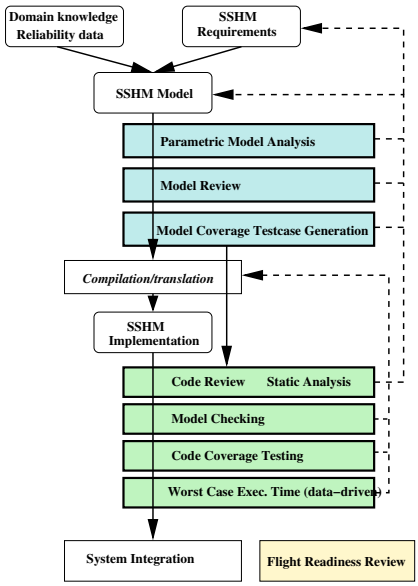

Fig. 6. V&V activities for an SSHM system; both on the model and implementation levels.

A. Model-level V&V

An SSHM model captures essential information about nominal and off-nominal operation of the host system on various levels of abstraction and is used by the SSHM engine to perform reasoning. Thus, V&V has to make sure that the model is *adequate* for the given domain and SSHM requirements and that it is as *complete* and *consistent* as possible. State-of-the-art V&V approaches include exhaustive model enumeration using a model checker, for example, Livingston Pathfinder [15]. Larger and hybrid models can be tested using parametric testing. This statistical approach combines n-factor combinatorial exploration with advanced data analysis [16] to exercise the SSHM model with wide ranges of sensor inputs and internal parameters. This approach scales to large systems, like fault detection models for the Ares I rocket [17] or hardware Health Management systems [18]. Finally, sensitivity analysis for Bayesian Networks [4] is useful to assess the quality of the model parameters. The results of this analysis provide arguments about the quality of the model and its expected behavior.

B. Code-level V&V

Since the health management functions are performed by software, this software—even if the model has been verified—has to undergo rigorous V&V ("code-level V&V"). Here, the SSHM is treated like a regular piece of software, which has to be tested and validated. In most cases, this will include testing according to established code coverage criteria, e.g., the MCDC (Modified Condition Decision Coverage) as required

by the DO-178B standard [19], worst case execution time analysis, stack and memory analysis, etc.

Typically, BN model compilation (e.g., into arithmetic circuits [4]) eliminates the problems often associated with complex reasoning algorithms associated with BNs; the resulting algorithms and data structures can be shown to have limited resource bounds and do not require dynamic memory allocation. The minimalistic SSHM reasoning engines that are the target of model translation and compilation might even be amenable to formal verification. Because of the high complexity of the compilation process, however, it is hard to provide any guarantees about the compiler implementation. In that case, techniques known from the area of compilers, like proof carrying code [20], might apply.

In all cases, the SSHM takes inputs from many different sources (hardware, sensors, software sensors, operating system, etc.) and thus has to interact with the host system as well as other software systems on multiple levels. The resulting SSHM architecture can therefore become rather complicated and requires careful V&V. Research has been performed to develop architectures that provide specific capabilities for SSHM integration [21].

Once the BN has been compiled into an arithmetic circuit, a small evaluator reads the commands sensor signals, and computes the health information. By construction, all data structures are static, there is no nondeterminism, and the execution times are bounded. Many V&V techniques that have been developed for traditional software can be used or extended easily. In particular, software model checking for the automatic proof of safety properties, static analysis, automatic generation of test cases and worst-case execution time analysis seem to be suitable approaches to demonstrate software safety and reliability.

We have to note that SSHM can not (and should not) lessen the burden of V&V and certification of the entire software. It only provides an additional layer of protection during run-time, as it is capable of detecting and identifying faults that have not been found during V&V or that are caused by unexpected environmental circumstances. SSHM does not intend to replace system V&V.

V. Related Work

System health management, including diagnosis, has been investigated by several other researchers [4], [22]–[31]. We now briefly discuss some closely related work.

Using the Bayesian approach [3], [4], some research on diagnosis using hybrid BNs already exists [25], [32]. Furthermore, two main approaches to handling hybrid behavior in a discrete probabilistic setting have been discussed: discretization [33], [34] and uncertain evidence, in particular soft evidence [3], [4], [35]. Discretization can be performed dynamically on-line [33], [34], however this is not done in this paper since it is inconsistent with our focus on off-line compilation into arithmetic circuits for fast (on-board) inference.

The present paper, which builds on previous hardware-oriented research on system health management using BNs [5], [7], [36], is in several ways different from previous work. First, we do not make the single-fault assumption. Second, we support a broad range of fault types, and in particular integrate software and hardware faults. Third, the compilation approach [37], [38] sets it apart from many other diagnostic systems (including Livingstone, L2, and HyDE and related systems [24], [29]). Fourth, it is an exact approach that does not rely on approximations, as do, for example, particle filters. Fifth, we do not use a dynamic BN [25], [28], just a simpler static BN with temporal evidence (see also [5], [7], [36]).

VI. Conclusions

In this paper, we have demonstrated how a Software and Sensor Health Management (SSHM) system can help to perform on-board fault detection and diagnosis for a small satellite. We have used Bayesian networks as the underlying modeling paradigm to concisely capture and merge information from hardware sensors, software status signals, software quality signals, and information from the OS. Given these data, Bayesian reasoning can compute the most likely causes of failures, if present, and also give a statistically sound measure for the quality (probability) of the answer.

System health models in the form of Bayesian networks can be compiled into efficient arithmetic circuits, which yield a high-performance SSHM and are suitable for execution within embedded (on-board) software systems, and are amenable to V&V. Furthermore, software tools for Bayesian modeling and compilation into arithmetic circuits—such as SamIam and Ace[6]—are readily available. Only if the SSHM proves to work reliably and without interference, such software components will be accepted on-board a small satellite.

We have illustrated this technique with a highly simplistic satellite GN&C system and discussed several (injected) failure scenarios where the SSHM is able to detect and diagnose software and hardware problems. We have set up a demonstrator using OSEK as the underlying operating system. Despite its simplicity, OSEK is used in many automotive and industrial applications. The entire software including the spacecraft simulator is running under Trampoline[7] on a Mac with OSX. In this paper, we have demonstrated that an SSHM system can work in such a restricted embedded environment.

However, this example only demonstrates a small range of the SSHM's capabilities. Current work investigates, how information on the quality of a computation (e.g., numerical quality or quality of the state estimation) can be smoothly incorporated into the SSHM. Research on hierarchical SSHMs will address the issue of detecting complicated software-hardware interactions for large- and extreme-scale BNs [39], and will focus on the fusion of multiple information streams for the purpose of increasing diagnostic accuracy. Finally, future work will investigate how our SSHM approach can deal

[6]http://reasoning.cs.ucla.edu/ace/

[7]http://trampoline.rts-software.org

with unexpected and unmodeled failures (e.g., due to unforeseen environmental circumstances) and emerging behavior.

SSHM has, due to its modeling capabilities, its efficient execution, and high reasoning power, the ability to replace current simplistic diagnostic code on-board even small spacecraft.

ACKNOWLEDGMENT

This work is supported by a NASA NRA grant NNX08AY50A "ISWHM: Tools and Techniques for Software and System Health Management". The third author was in part supported by the NASA USRP internship program.

REFERENCES

[1] P. Wilhide, "Mars Program Assessment Report Outlines Route to Success," 2000. [Online]. Available: http://mars.jpl.nasa.gov/msp98/news/news71.html

[2] M. Adler, "The Planetary Society Blog: Spirit Sol 18 Anomaly," 2006. [Online]. Available: http://www.planetary.org/blog/article/00000702/

[3] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[4] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[5] B. W. Ricks and O. J. Mengshoel, "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, 2009.

[6] ——, "Diagnosing intermittent and persistent faults using static Bayesian networks," in *Proc. of the 21st International Workshop on Principles of Diagnosis (DX-10)*, Portland, OR, 2010.

[7] O. J. Mengshoel, M. Chavira, K. Cascio, S. Poll, A. Darwiche, and S. Uckun, "Probabilistic model-based diagnosis: An electrical power system case study," Systems, Man and Cybernetics, Vol. 40, No. 5, pp. 874–885, 2010.

[8] "History's worst software bugs," Wired.com, 2009.

[9] 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 *Proceedings of the Twentieth Innovative Applications of Artificial Intelligence Conference (IAAI-08)*, Chicago, IL, 2008, pp. 1699–1705.

[10] O. J. Mengshoel, A. Darwiche, and S. Uckun, "Sensor validation using Bayesian networks," in *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS-08)*, 2008.

[11] O. J. Mengshoel, "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)*, Nashville, TN, 2007, pp. 330–337.

[12] ——, "Understanding the scalability of Bayesian network inference using Clique Tree Growth Curves," *Artificial Intelligence*, vol. 174, pp. 984–1006, 2010.

[13] O. J. Mengshoel, S. Poll, and T. Kurtoglu, "Developing large-scale Bayesian networks by composition: Fault diagnosis of electrical power systems in aircraft and spacecraft," in *Proc. of the IJCAI-09 Workshop on Self-⋆ and Autonomous Systems (SAS): Reasoning and Integration Challenges*, 2009.

[14] W. River, "ARINC 653– an Avionics Standard for safe, partitioned Systems," IEEE Seminar, 2008. [Online]. Available: http://www.computersociety.it/wp-content/uploads/2008/08/ieee-cc-arinc6%53_final.pdf

[15] A. E. Lindsey and C. Pecheur, "Simulation-based Verification of autonomous Controllers via Livingstone Pathfinder," in *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004*. Lecture Notes in Computer Science, vol. 2988. Springer, 2004, pp. 357–371.

[16] J. Schumann, K. Gundy-Burlet, C. Pasareanu, T. Menzies, and T. Barrett, "Software V&V support by parametric analysis of large software simulation systems," in *Proc. IEEE Aerospace*. IEEE Press, 2009.

[17] J. Schumann, A. Bajwa, and P. Berg, "Parametric testing of launch vehicle FDDR models," in *AIAA Space*, 2010.

[18] E. Reed, J. Schumann, and O. J. Mengshoel, "Verification and Validation of System Health Management Models using Parametric Testing," in *Proc. Infotech@Aerospace*, 2011.

[19] "DO-178B: Software considerations in airborne systems and equipment certification," URL: `http://www.rtca.org`, 1992.

[20] G. C. Necula, "Proof-carrying code," in *Proc. 24th ACM Symp. Principles of Programming Languages*. Paris, France: ACM Press, Jan. 15–17 1997, pp. 106–19.

[21] 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.

[22] J. de Kleer and B. C. Williams, "Diagnosing multiple faults," *Artificial Intelligence*, vol. 32, no. 1, pp. 97–130, 1987.

[23] M. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper, "Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR knowledge base: I. The probabilistic model and inference algorithms," *Methods of Information in Medicine*, vol. 30, no. 4, pp. 241–255, 1991.

[24] B. C. Williams and U. Nayak, "A model-based approach to reactive self-configuring systems," in *In Proceedings of AAAI-96*, 1996, pp. 971–978.

[25] U. Lerner, R. Parr, D. Koller, and G. Biswas, "Bayesian fault detection and diagnosis in dynamic systems," in *Proceedings of the Seventeenth national Conference on Artificial Intelligence (AAAI-00)*, 2000, pp. 531–537.

[26] I. Rish, M. Brodie, and S. Ma, "Accuracy vs. efficiency trade-offs in probabilistic diagnosis," in *Eighteenth national conference on Artificial intelligence (AAAI-02)*, Edmonton, Canada, 2002, pp. 560–566.

[27] M.-O. Cordier, P. Dague, F. Levy, J. Montmain, M. Staroswiecki, and L. Trave-Massuyes, "Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, no. 5, pp. 2163–2177, 2004.

[28] I. Roychoudhury, G. Biswas, and X. Koutsoukos, "A Bayesian approach to efficient diagnosis of incipient faults," in *Proceedings of the 17th International Workshop on Principles of Diagnosis (DX-06)*, Spain, 2006, pp. 243–250.

[29] S. Narasimhan and L. Brownston, "Hyde: a general framework for stochastic and hybrid model-based diagnosis," in *Proc. 18th International Workshop on Principles of Diagnosis (DX-07)*, 2007, pp. 162–169.

[30] S. Ruan, Y. Zhou, F. Yu, K. R. Pattipati, P. Willett, and A. Patterson-Hine, "Dynamic multiple-fault diagnosis with imperfect tests," *IEEE Trans. Sys. Man Cyber. Part A*, vol. 39, no. 6, pp. 1224–1236, 2009.

[31] A. Srivastava and J. Han, Eds., *Data Mining in Systems Health Management: Detection, Diagnostics, and Prognostics*. Chapman and Hall/CRC Press, 2011.

[32] A. Choi, A. Darwiche, L. Zheng, and O. J. Mengshoel, "A Tutorial on Bayesian Networks for System Health Management." In [31], 2011.

[33] A. Kozlov and D. Koller, "Nonuniform dynamic discretization in hybrid networks," in *In Proc. UAI*. Morgan Kaufmann, 1997, pp. 314–325.

[34] H. Langseth, T. D. Nielsen, R. Rumi, and A. Salmeron, "Inference in hybrid bayesian networks," *Reliability Engineering & System Safety*, vol. 94, no. 10, pp. 1499–1509, 2009.

[35] H. Chan and A. Darwiche, "On the revision of probabilistic beliefs using uncertain evidence," *Artificial Intelligence*, vol. 163, no. 1, pp. 67–90, 2005.

[36] B. W. Ricks and O. J. Mengshoel, "The diagnostic challenge competition: Probabilistic techniques for fault diagnosis in electrical power systems," in *Proc. of the 20th International Workshop on Principles of Diagnosis (DX-09)*, Stockholm, Sweden, 2009.

[37] A. Darwiche, "A differential approach to inference in Bayesian networks," *Journal of the ACM*, vol. 50, no. 3, pp. 280–305, 2003.

[38] M. Chavira and A. Darwiche, "Compiling Bayesian networks using variable elimination," in *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India, 2007, pp. 2443–2449.

[39] K. Przytula, G. Isdale, and T.-S. Lu, "Collaborative development of large Bayesian networks," in *Proc. of the 2006 IEEE Autotestcon*, 2006, pp. 515–522.