NASA/CR–2013-217978

Vehicle Integrated Prognostic Reasoner (VIPR) Metric Report

Dennis Cornhill, Raj Bharadwaj, and Dinkar Mylaraswamy
Honeywell International, Inc., Golden Valley, Minnesota

April 2013

NASA/CR–2013-217978

Vehicle Integrated Prognostic Reasoner (VIPR) Metric Report

Dennis Cornhill, Raj Bharadwaj, and Dinkar Mylaraswamy
Honeywell International, Inc., Golden Valley, Minnesota

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NNL09AD44T

April 2013

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Contents

Table of Figures

1 Introduction

This document outlines a set of metrics for evaluating the diagnostic and prognostic schemes developed for the Vehicle Integrated Prognostic Reasoner (VIPR). A number of diagnostic and prognostic metrics are defined in the literature (e.g., [1, 2, 3]), but these standards are defined for well-circumscribed algorithms that apply to small subsystems. VIPR is designed to be a system-level reasoner that encompasses multiple levels of large, complex systems such as those for aircraft and spacecraft. The wide variety of reasoners employed in such systems span from individual line replaceable unit (LRU) health managers to area health managers (AHM) and the vehicle health manager (VHM) [4]. These health managers are organized hierarchically and operate together to derive diagnostic and prognostic inferences from symptoms and conditions reported by a set of diagnostic and prognostic monitors (DMs and PMs) [5]. A brief description of the layered VIPR architecture is presented in Section 2 of this document.

Existing metrics for evaluating fault detection, fault isolation, and prognostics schemes are directly applicable to the DMs, PMs, and LRU health managers. For layered reasoners such as VIPR, the overall performance cannot be evaluated by metrics solely directed toward timely detection and accuracy of estimation of the faults in individual components. Among other factors, overall vehicle reasoner performance is governed by the effectiveness of the communication schemes between monitors and reasoners in the architecture, and the ability to propagate and fuse relevant information to make accurate, consistent, and timely predictions at different levels of the reasoner hierarchy.

To address these issues, we outline an extended set of diagnostic and prognostics metrics in this report. These metrics can be broadly categorized as evaluation measures for: (1) diagnostic coverage, (2) prognostic coverage, (3) accuracy of inferences, (4) latency in making inferences, and (5) sensitivity to different fault and degradation conditions. We also discuss possible cost-benefit metrics to capture the improved performance-to-cost calculations for the VIPR layered architecture.

Our overall approach involves a systematic study of the effectiveness of the VIPR system using a simulation testbed designed to generate off-nominal events corresponding to several fault scenarios [5]. A set of these fault scenarios is documented in a previous report [5]. The evaluation studies involve systematic generation of degradation and fault data, realistic analysis using the monitors and reasoners in the VIPR architecture, and a methodology to compute the values for the chosen metrics using the performance data collected from the software testbed. Benchmarking VIPR makes it possible to assess how it can increase aviation safety. To achieve a benchmark, we evaluate VIPR's performance using the diagnostics and prognostics metrics described in Section 4.

Section 7 discusses the Hardware-in-the-Loop demonstration and reports on additional metrics extracted from a VIPR demo and test configuration that now includes a piece of avionics equipment, Honeywell's LaserRef VI Inertial Reference Unit (IRU).

The rest of this report outlines the VIPR architecture, the simulation testbed for benchmarking studies, the metrics chosen for diagnostic and prognostics reasoner evaluation, and the summary and conclusions for this report.

THIS PAGE INTENTIONALLY LEFT BLANK

2 VIPR Architecture and Functionality

The primary function of VIPR is to detect and isolate faults and failures at the aircraft level. A simplified functional view of VIPR is shown in Figure 1. VIPR is organized into a hierarchical architecture. At each level or layer of the hierarchy, the VIPR processing blocks maintain relationships with other blocks at the same level.

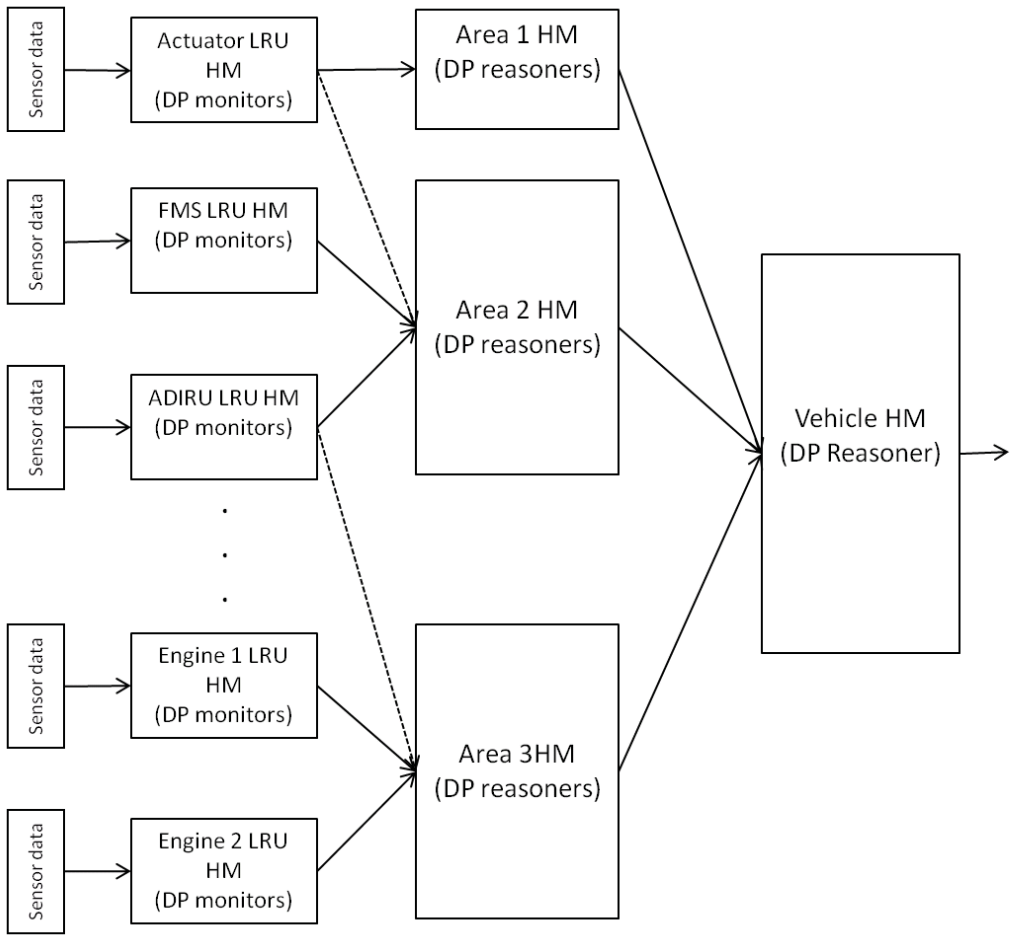


Figure 1: Functional View of VIPR

To satisfy bandwidth and power constraints [4, 5], only a subset of messages is allowed to flow from one level to another. At the lower level LRU, HMs receive measurements from the sensors, and they perform diagnostic and prognostic (DP) monitoring tasks to compute DP indicators. The next level is organized into multiple AHMs that follow the principal spatial and temporal decomposition of the aircraft functionality and behavior. The main task of an area HM is to perform DP reasoning using the indicators provided by the LRU HMs. Finally, a VHM is responsible for collecting the data from all AHMs and solving any ambiguities with the assistance, if necessary, of off-vehicle health management services. In the following, we first describe the candidate algorithms that can be used for the VIPR and then discuss the information flow between the various levels of the VIPR.

LRU Health Manager

At the LRU level, the objective is twofold: (1) discover information that can be used for DP reasoning in raw and noisy measurements by performing feature extraction and (2) compress the data so that they can be efficiently transmitted and used by the higher levels for more integrated analyses (e.g., reasoning about the effects of fault propagation between subsystems). Both tasks are accomplished by a suite of DP monitors that can be classified into two categories: (1) simple DP monitors and (2) advanced DP monitors.

Simple DP monitors test whether a sensor measurement or measurement rate exceeds a threshold. All major subsystems in an aircraft have built-in tests (BIT) that perform such operations and present the simplest form of feature extraction, generating binary health indicators. Mathematically, these tests are based on well-defined detectors such as likelihood ratio test, z-test, and t-test. In addition to such algorithms, advanced DP monitors are used for discovering and extracting information from multivariable measurement sets.

A representative algorithm for such a monitor is principal component analysis (PCA), which transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. After using multivariate signal processing algorithms, advanced DP monitors can use simple classification and trending algorithms to encapsulate information to DP indicators that are forwarded to the Area HMs. Candidate algorithms include bin classifiers, nearest-neighbor classifiers, and discriminant analysis. The computed indicators include: (1) condition indicators that can describe, for example, the engine compressor efficiency and spectral energy content from a vibration signal; (2) health indicators that capture, for example, inlet filter, compressor rub, or foreign object damage (FOD) incidents; and (3) prognostic indicators that show, for example, the evolution of engine health for the next 100 hours of a specific mission.

Area Health Manager

AHMs conduct DP reasoning for aircraft subsystems, including multiple LRUs. They are organized along spatial and temporal boundaries of fault manifestation and propagation to minimize the communication between aircraft subsystems. Since perfect containment of a fault in one area cannot be guaranteed, AHMs can query remote LRUs if necessary. Candidate algorithms at this level include decision trees, discrete event system diagnosers, failure propagation graphs, neural networks, fuzzy classifiers, and Bayesian networks. Heterogeneous reasoners deal with binary, discrete, and continuous indicators provided by the LRUs as well as with event-driven and time-driven dynamics of the underlying aircraft components.

Vehicle Health Manager

The VHM is responsible for reasoning across spatial and temporal boundaries of the various areas and possibly uses off-vehicle health management services. The VHM resolves ambiguities that may arise from the AHMs, initiates additional DP tests, and provides warnings. The DP reasoning technologies are similar to those in the AHMs, but special care must be taken to deal with multiple temporal scales.

Reference Models

Managing and evaluating the operation of VIPR requires a database of the health management components that are available at the LRU, area, and vehicle levels. Every component is associated with a reference model (implemented as an XML file) that captures the interface of the component and information about the internal functionality if available. At the very least, it defines the input output relations for the associated component.

Figure 2 and Figure 3 show hierarchical and central/flat reasoning. Both reasoners use corresponding reference models that define the monitors source and diagnostic relationships. In the hierarchical reference model, the LRU reference model defines all the monitors at the LRU level as well as the bipartite graph that connects the monitors to the LRU failure modes. The fault cascade/common causes are captured at the area level along with the faults in area level systems such as the fuel lines, electrical buses, etc. At the vehicle level, the reasoner model captures the relationships in between the area-level faults and defines the inhibitors to suppress nuisance faults.

VIPR uses a reference model designed to support a VIPR-type hierarchical system. In such a system, monitor generation and active queries initiation resolve the LRU-level ambiguity. The area-level reasoner uses the area-level reference model to isolate faults and discover fault cascades. In contrast, the central reasoners use a flat reference model. In this type of reasoning, the monitors may be generated at LRU level. All other functions of the reasoner, including query, isolation, cascade reasoning, fusion, and inhibits are performed at the vehicle level. The VIPR reasoner code is designed to use either reference model for reasoning, which allows comparison of results from the distributed (hierarchical) and central (flat) reasoners.

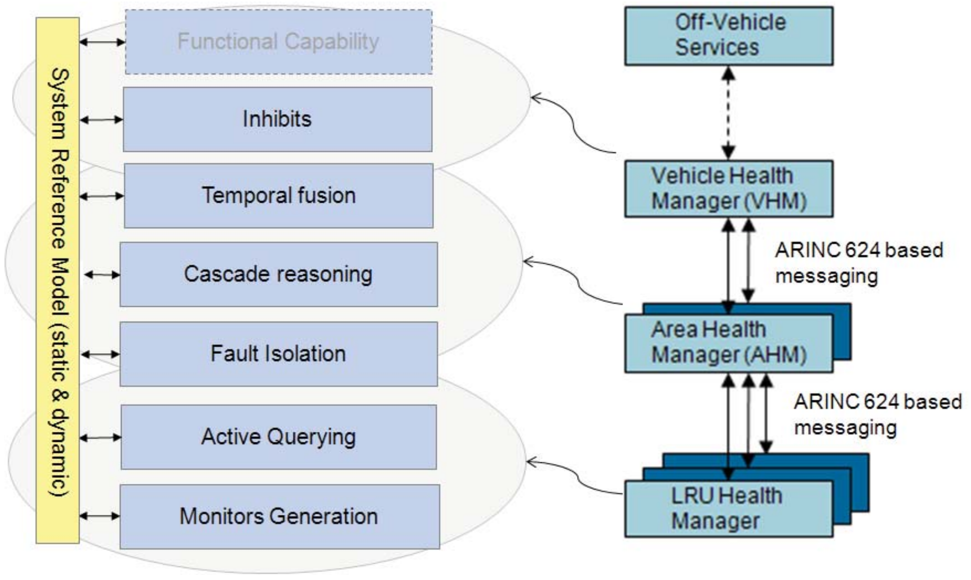


Figure 2: Hierarchical Reasoning

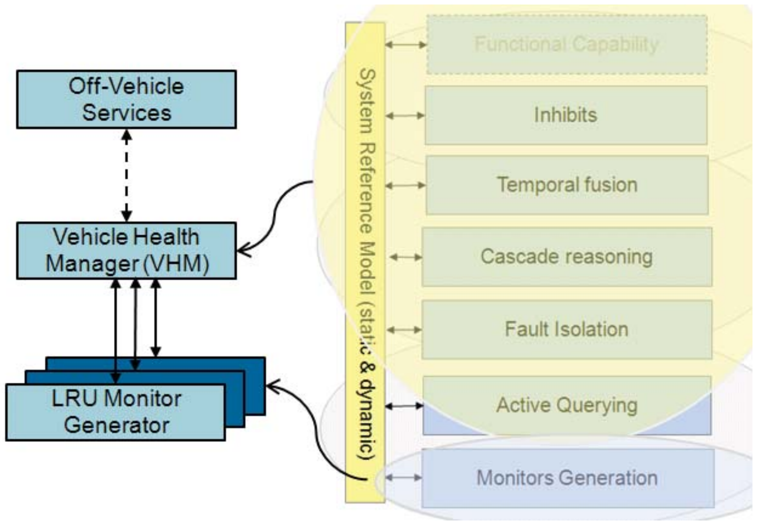


Figure 3: Central Reasoning

Information Flow

VIPR's basic information flow starts from the sensors that communicate raw measurements to the LRU HMs. The LRU HMs compute DP indicators using simple or advanced DP monitors and send them to the AHMs. The DP reasoners in the AHMs generate fault candidates that are sent to the VHM. At the vehicle level, reasoners generate detections and predictions of failure modes and advisories.

In addition to communicating the output of the health management components at this level, VIPR considers an enhanced information flow that complements the component results with metadata that provide valuable information related to how these results have been computed. The metadata communicated instantiate the attributes of the reference model of the corresponding component in the VIPR architecture constructing an accurate runtime representation of the VIPR configuration.

The information flow can then follow two paradigms. First, low-level components can forward important messages to higher levels upon detection, for example, of adverse events. Second, high-level components can actively query low-level components for specific information that can be used to disambiguate fault candidates or improve fault prediction. In addition, VIPR supports active sensor tests that are invoked on demand.

Evaluation

Evaluation of the vehicle-level health management architecture, such as VIPR, must assess how it can increase aviation safety. This goal is directly linked to the following measures:

1. Diagnostic coverage
2. Prognostic coverage
3. Accuracy
4. Latency
5. Sensitivity

Given VIPR's hierarchical architecture, the benchmarking process must consist of two steps:

1. Quantifying the effectiveness of each VIPR in terms of the above metrics

2. Determining the accumulated inaccuracies as information is passed up the architecture

Well-defined metrics that can be used to evaluate VIPR performance (see next section). In addition to such measures, it is important to evaluate the efficiency and scalability of VIPR in terms of the computational resources needed as well as to quantify the trade-offs between performance and resource usage. The enhanced information flow and the active querying described above, for example, can improve performance and increase safety but they require increased computation and communication capabilities. Cost-benefit analysis is then necessary to determine the optimal VIPR configuration. The following section describes the metrics to be computed along with the discussion on the cost analysis.

3 VIPR Metrics

This section defines what and how we generate summary statistics from the algorithm performance and message logs and describes the Monte Carlo experiments run to exercise the reasoner and generate performance and message logs.

A number of diagnostic and prognostic metrics exist, but these standards are defined for well-circumscribed algorithms that apply to small subsystems. For layered reasoners such as VIPR, the overall performance cannot be evaluated by metrics directed solely toward timely detection and accuracy of estimation of the faults in individual components. Among other factors, the overall vehicle reasoner performance is governed by the effectiveness of the communication schemes between monitors and hierarchical reasoners and the ability to propagate and fuse relevant information to make accurate, consistent, and timely predictions at different levels of the reasoner hierarchy. An added functionality of this architecture is the ability of the vehicle- and area-level reasoners to generate specific queries for the component monitors. To address these issues, we have developed an extended set of diagnostic and prognostics metrics that can be used to evaluate the performance of the layered architecture. The metrics are summarized in the following sections.

3.1 Accuracy and Computation Metrics

3.1.1 Accuracy Metrics

Generation of the reasoner accuracy metrics: The reasoner accuracy metrics are generated by running the reasoner in tandem with the fault simulator. The fault simulator works as an evidence source that generates the monitors for the seeded faults. The reasoner accuracy metrics are captured directly from the MATLAB® run. It is not necessary to run the GUI to capture the reasoner accuracy metrics; however, the reasoner accuracy metrics are calculated for all runs, including the non-metrics collection runs from the GUI and are displayed on the MATLAB console at the end of the reasoner run.

The following data is collected to capture the reasoner accuracy metrics:

- Simulated fault is captured for metrics computation.
- List of all monitors that fired and the time at first firing.
- Diagnostic accuracy, which is the accuracy of the reasoner's diagnostic conclusions. It is accomplished by comparing the final reasoner conclusion for all the simulated faults. The diagnostics accuracy captures the following sub-metrics:

    - Rate to false alarms: given absence of faults, detecting faults when no faults are present.
    - Rate of true detects: given presence of detected faults, detecting the faults that are present.
    - Rate of false detects: given presence of some faults, detecting faults that are not present/simulated in the scenario.
    - Rate of miss detects: given the presence of some faults, missing the detection of a simulated fault.
- Prognostic accuracy: Accuracy of the prognostics. The metrics captures the fusion of two or more prognostic vectors and discovery of precursors through data mining
- Time to detect as measured as time from the first appearance of the indicting monitor.
- Time to isolate two or more faults. This is also measured from the time of initiation of the first set of monitors corresponding to the simulated faults.
- Detection rate for intermittent faults. The types of intermittent fault are shown in Figure 4. For accuracy metrics, we concentrate on the intermittent evidence leg.

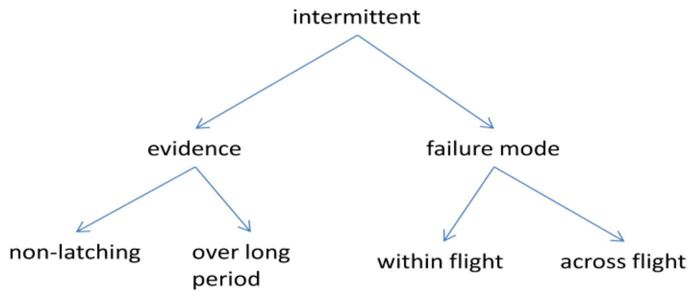


Figure 4: Types of intermittent faults

- Isolation layer/reasoner: This metric captures information about the reasoner that isolated the fault for all simulated faults. The conjecture is that complex faults are isolated by higher level reasoners (such as the AHM, VHM reasoner) and, this metric will verify that hypothesis.

This study is repeated for a select set of faults with multiple reasoner parameters, such as:

- Threshold for splitting/merging
- Threshold for acceptance/rejection
- Threshold for isolation/ambiguous
- Threshold for fault condition closeout

3.1.2 Computational Metrics

Communication and profiling data is collected at run time to compute the complexity cost. The communication metrics are utilized to compare the distributed layered reasoner architecture with the central reasoner architecture. In the distributed reasoner, the computations occur at all the layers and only the conclusions, active queries, and broadcasts are sent out to the next reasoner level. In the central reasoner, all the computations and disambiguation occurs at the central reasoner.

To better understand the communication tradeoffs the following data is collected:

- Communication costs: Communication costs are computed from the total number of communications and the bandwidth utilization required to isolate a fault by the reasoner in a given architecture. Computation begins at the time of fault inception. To compute the communication costs, these data items need to be logged:
    - Message source ID
    - Destination ID
    - Timestamp
    - Message number
    - Packet type
    - Packet subtype
    - Payload size

  Message cost can be described as by cost function that is proportional to the payload size from source to destination layers.
- Message delays can be incorporated in the communication cost through post analysis. For example, for a message from an LRU reasoner to the AHM, the communication delay could be modeled with a bounded delay.
- Bandwidth utilization computation is accomplished by assuming that VIPR has a fixed maximum percent of the communication bandwidth and then computing the latency implied by the bandwidth limitation. For example, the communication bandwidth can be assumed to be 10Mbps, and VIPR can be assumed to be limited to, at most, 1% of the communication bandwidth. The bandwidth utilization computation compares the communications in both the central and distributed reasoner architectures.
- At the reasoner level, the following information is also collected:
    - CPU execution times
    - CPU utilization
    - Memory utilization

3.2 Monte Carlo Experiments

We use Monte Carlo experiments to exercise the reasoner and generate performance and message logs. The objective of these experiments is to generate the:

- Accuracy metrics
- Computation metrics

The accuracy metrics are generated by running the reasoner using the complete VIPR reference model along with the fault simulator under the following combinations of conditions:

- Faults number:
    - No fault baseline
    - All possible single complex faults
    - All combinations of two complex fault conditions
- Fault types

    - Latched faults
- Evidence/Monitors
    - Random evidence coming on line, i.e. the monitors fire based on stochastic probabilities
- Evidence type:
    - Latched monitors
    - Chattering monitors (only for non-indicting monitors)
- Special cases:
    - Two or more prognostic monitors need to be fused to generate the accuracy metrics
    - Multiple fault cases that lead to ambiguity in the reasoner conclusions
- Time of inception of fault
    - 200 seconds into flight
- Reasoner parameters: Repeated simulations for the selected set of faults with multiple reasoner parameters, such as:
    - Threshold for splitting (3 thresholds)
    - Threshold for merging (3 thresholds)

The computational metrics are run over all faults for the small reference model with both flattened and layered reference models. We consider all 1-fault and 2-fault combinations and study only latched fault and monitor states. The effect of simultaneous-versus-staged evidence is also studied. The communications costs are logged and analyzed further by fitting multiple communication delay/cost models.

4 VIPR Evaluation Approach

The VIPR evaluation approach is illustrated in Figure 5. We used a regional airline data base to enhance the reference model and to generate monitors to test VIPR. The reference model is also an input parameter to the failure mode simulator. The failure mode simulator generates evidence streams that correspond to a selected failure mode from the reference model. The evidence stream is then fed to the VIPR reasoner. Reasoner outputs such as isolated faults, detected faults, time of isolation, isolating reasoner, etc. are logged and analyzed by the metrics analysis scripts. The metrics analysis scripts summarize the results and calculate false alarm rates, true detect rates, average time to isolate, number of reasoner messages, etc., which is then used for reasoner and reference model improvements.

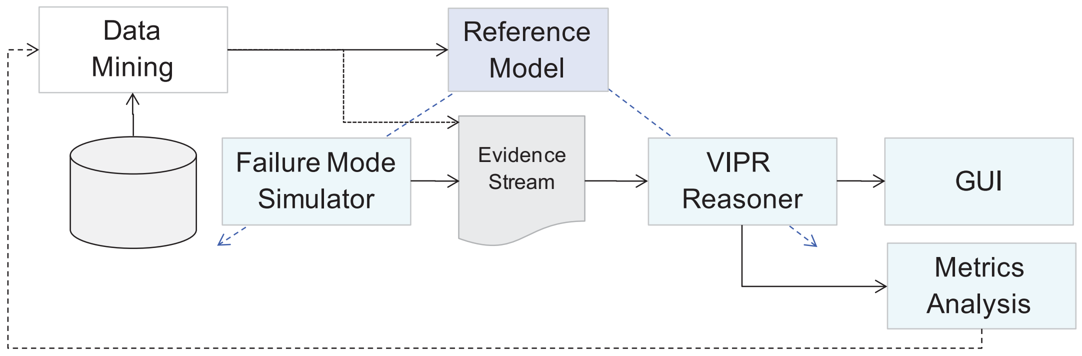


Figure 5: VIPR Evaluation Approach

Figure 6 shows four places in the overall system flow of information where inputs to the reasoning process can greatly affect the quality of the reasoner's results.

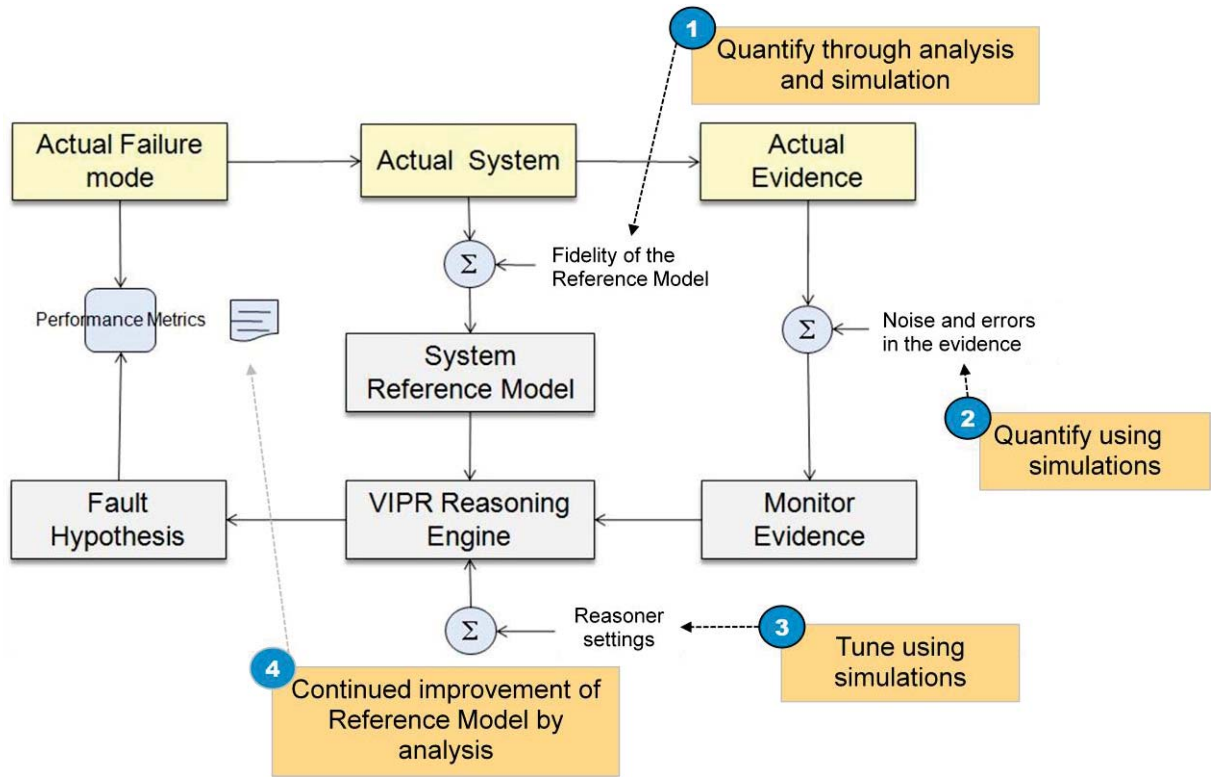


Figure 6: VIPR Data Flow Showing Four Opportunities for Evaluating Results and Refining Inputs

1 is the reference model for describing the vehicle—clearly, it needs to be an accurate reflection of the vehicle.

2 is the quality of the evidence stream—low-fidelity evidence will erode the reasoner's ability to accurately isolate a fault.

3 indicates the settings that tune the reasoner's operation, such as the confidence threshold for declaring a fault isolated.

4 measures the reasoner's effectiveness and use those results to fine tune the reference model.

We evaluated the sensitivity of the reasoner settings and reported results in Section 6.2.2. The other three evaluations need to be done in the context of a specific vehicle, high-fidelity reference model, and actual evidence streams.

5 Failure Mode Simulator

We developed the failure mode simulator to enable exhaustive testing of the VIPR reasoner because it is not possible to extract all types of failure modes from the regional airline data base. The failure mode simulator uses the reference model to generate the monitor evidence for the simulated faults. The simulator uses stochastic processes for setting monitor firing and monitor activation times, and for generating false alarms.

Figure 7 and Figure 8 show the reference model in graphical and textual format. Figure 9 shows the three thresholds that are used to generate the diagnostic monitors. The three thresholds are used against uniformly distributed random numbers to simulate the stochastic nature of the monitors. The first threshold, the Monitor Valid threshold, is an exponential threshold that captures the behavior of monitors as they come on-line at the beginning of the flight. This threshold is used to generate an invalid/cannot compute (represented as -1) monitor response. The false alarm threshold is used to generate false monitor firing throughout the simulation window. The fault detect threshold becomes active only after the injection of fault. It is used to decide if the corresponding monitor fires on/or not when the fault is present (injected). Both the false alarm and the detection threshold can be read off the reference model as highlighted in Figure 7.

Following steps capture working of the fault simulator and monitor /evidence generation. It is assumed that the simulated failure mode is represented in the reference model. Then:

1. Find all monitors (*M*) in the reference model
2. Simulate an array of *M times T,* where *T* is the number of frames of evidence to be generated
3. The monitor generation program includes some additional logic that first determines the validation of the monitor. The validity is determined by checking if the simulated sample value is greater than the Monitor valid threshold. If this condition is not met then *Mj is set to -1*. If the monitor is valid then Monitor False alarm or the fault injection threshold is used to set the monitor *Mj = 0 or 1.*
4. After sample time >=fault injection time
    a. Get all evidence of interest (monitors) and their detection probability
    b. Find if there are any prognostic monitors associated with the diagnostics monitors. Simulate the time of issue for the prognostics monitors. Simulate the prognostics vector.
    c. For all indicting diagnostics monitor check if evidence of interest *Mj>1-dij,* then, *Mj = 1 otherwise 0*
5. Time *Mj* first becomes *>1-dij* is set as monitor time stamp

Figure 10 shows the sample diagnostics and prognostics vectors. Simulations are run multiple times using different random number seeds.

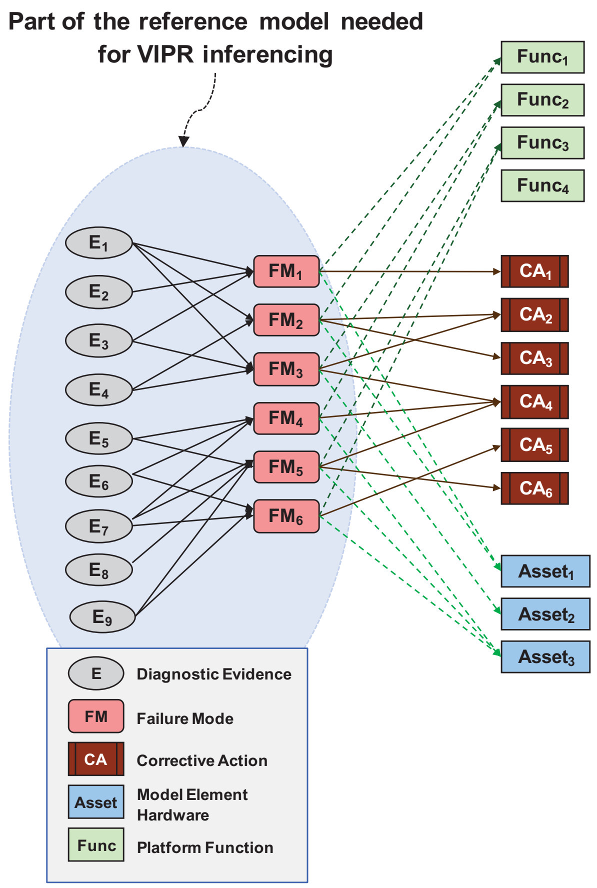
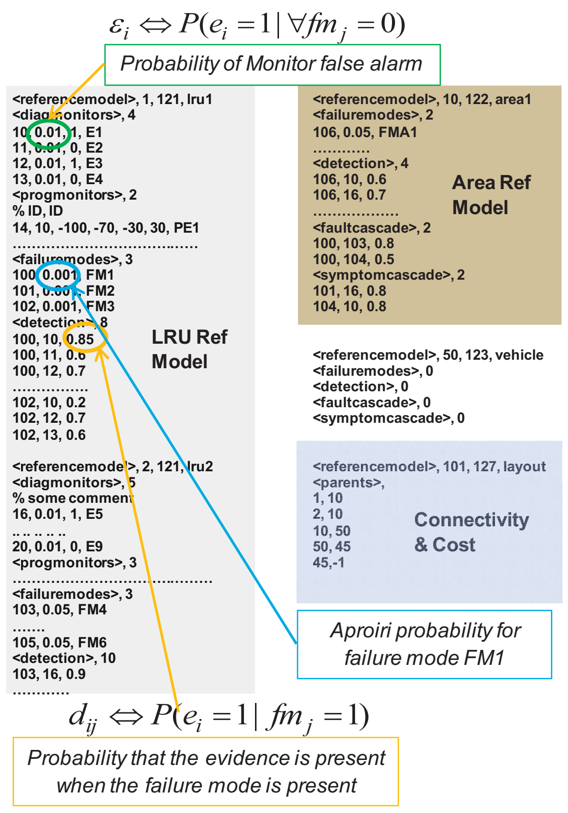


Figure 7: Graphical representation of the reference model


$$\varepsilon_i \Leftrightarrow P(e_i = 1 \mid \forall fm_j = 0)$$

Probability of Monitor false alarm

```
<referencemodel>, 1, 121, lru1
<diagmonitors>, 4
10, 0.01, 1, E1
11, 0.01, 0, E2
12, 0.01, 1, E3
13, 0.01, 0, E4
<progmonitors>, 2
% ID, ID
14, 10, -100, -70, -30, 30, PE1
…………………………………
<failuremodes>, 3
100, 0.001, FM1
101, 0.001, FM2
102, 0.001, FM3
<detection>, 8
100, 10, 0.85
100, 11, 0.6
100, 12, 0.7
……………….
102, 10, 0.2
102, 12, 0.7
102, 13, 0.6

<referencemodel>, 2, 121, lru2
<diagmonitors>, 5
% some comment
16, 0.01, 1, E5
.. .. .. .. ..
20, 0.01, 0, E9
<progmonitors>, 3
…………………………………………
<failuremodes>, 3
103, 0.05, FM4
…….
105, 0.05, FM6
<detection>, 10
103, 16, 0.9
…………
```

LRU Ref Model

```
<referencemodel>, 10, 122, area1
<failuremodes>, 2
106, 0.05, FMA1
…………
<detection>, 4
106, 10, 0.6
106, 16, 0.7
………………
<faultcascade>, 2
100, 103, 0.8
100, 104, 0.5
<symptomcascade>, 2
101, 16, 0.8
104, 10, 0.8
```

Area Ref Model

```
<referencemodel>, 50, 123, vehicle
<failuremodes>, 0
<detection>, 0
<faultcascade>, 0
<symptomcascade>, 0
```

```
<referencemodel>, 101, 127, layout
<parents>,
1, 10
2, 10
10, 50
50, 45
45,-1
```

Connectivity & Cost

Aproiri probability for failure mode FM1

$$d_{ij} \Leftrightarrow P(e_i = 1 \mid fm_j = 1)$$

Probability that the evidence is present when the failure mode is present

Figure 8: Reference model example

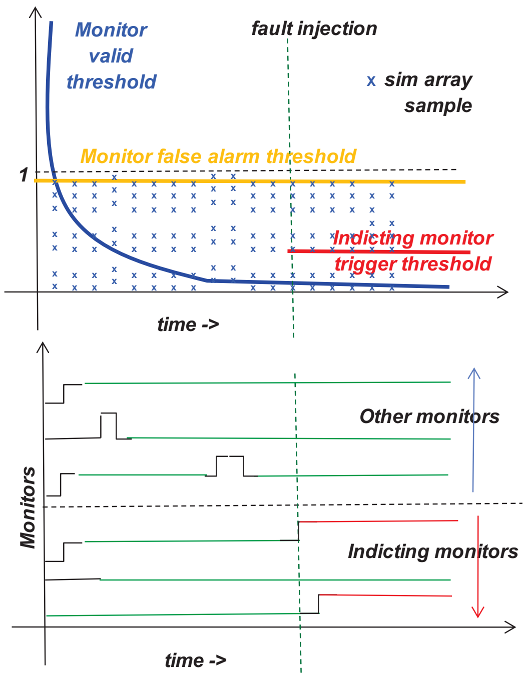
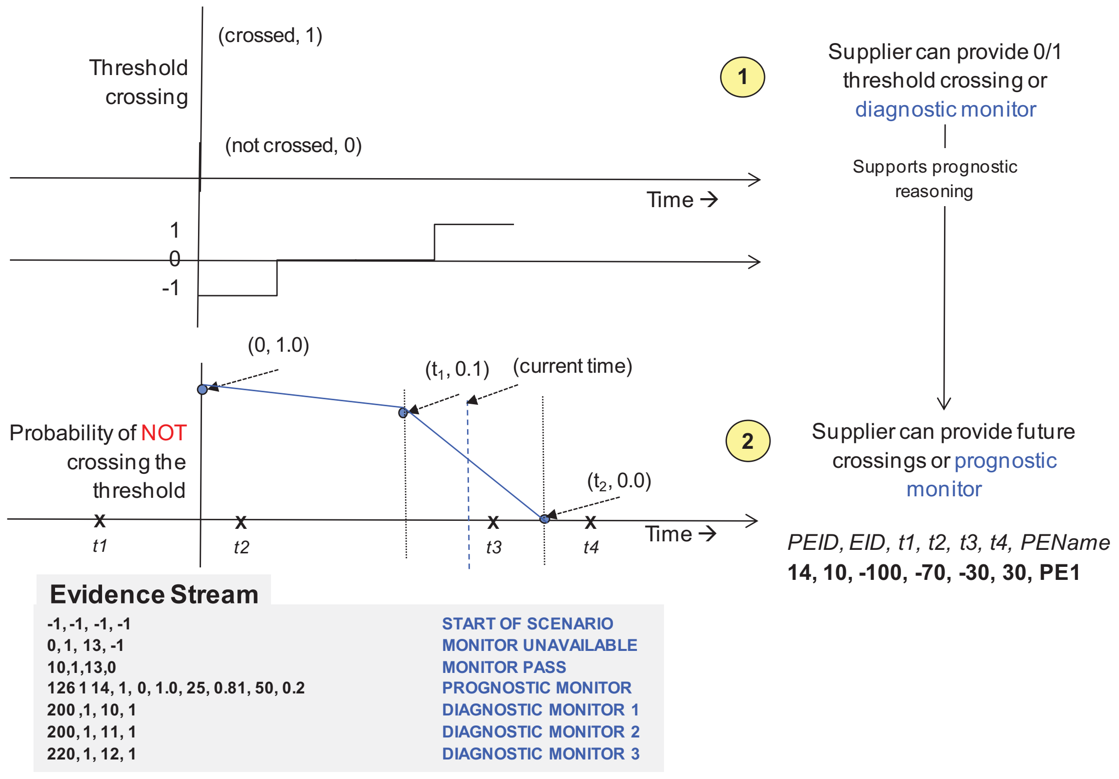


Figure 9: (a) Three thresholds for monitor generation; (b) Diagnostics monitors



Threshold crossing
(crossed, 1)
(not crossed, 0)
Time →
1
0
-1
1
2
Supplier can provide 0/1 threshold crossing or diagnostic monitor
Supports prognostic reasoning
Supplier can provide future crossings or prognostic monitor
(0, 1.0)
(t_1, 0.1)
(current time)
(t_2, 0.0)
Probability of NOT crossing the threshold
t1
t2
t3
t4
Time →

PEID, EID, t1, t2, t3, t4, PEName
14, 10, -100, -70, -30, 30, PE1

Evidence Stream

-1, -1, -1, -1	**START OF SCENARIO**
0, 1, 13, -1	**MONITOR UNAVAILABLE**
10,1,13,0	**MONITOR PASS**
126 1 14, 1, 0, 1.0, 25, 0.81, 50, 0.2	**PROGNOSTIC MONITOR**
200 ,1, 10, 1	**DIAGNOSTIC MONITOR 1**
200, 1, 11, 1	**DIAGNOSTIC MONITOR 2**
220, 1, 12, 1	**DIAGNOSTIC MONITOR 3**

Figure 10: Types of evidence generated by the failure mode simulator

5.1 Selection of Failure Modes for Simulation

The failure modes to be simulated by the simulator are selected only if they satisfy the following criteria for complex failure modes (Figure 11):

1. Select monitors that indict two or more failure modes. Monitors that are linked to a single fault have been eliminated because they do not generate interesting tests.
2. Select failure modes that have two or more indicting monitors.

The hierarchical and flat reference models each defined these 22 complex fault conditions:

- APU
    - EC Blade Rub
    - Fuel Metering Fault
    - Starter Fault
    - Igniter Assembly Fault
    - Turbine Erosion
    - Nozzle Clogging
    - ECB Fault
    - No Fuel
    - Bearing Fault
    - Inlet Blocked

- Engine
    - AI Stuck Valve
    - Fan/LPC Degradation
    - HPC Degradation
    - AC Duct Rupture
    - Fadec Fault
    - Fuel Metering Fault
    - HPT Degradation
    - Igniter Fault
    - Inlet Fouling
    - Shutoff Drain Valve Fault
    - Starter Fault
    - Nozzle Clogged

The simulations are configured to exercise only the complex failure modes, which ensures that the statistics are not skewed by the simple faults which are easy to detect and isolate.

	E1	E2	E3	E4	E5	E6	E7	SR
FM1	1	0	0	0	0	0	0	1
FM2	0	1	0	0	0	0	0	1
FM3	1	1	0	0	0	0	0	2
FM4	0	0	1	1	1	0	0	3
FM5	0	0	1	0	1	1	1	4
FM6	0	0	0	1	1	0	1	3
FM7	0	0	1	0	0	0	0	1
FM8	0	1	0	0	1	0	0	2
	2	3	3	2	4	1	2	

Figure 11: Failure mode to detection matrix

6 Profiling the Reasoner

6.1 Overview

Figure 12 illustrates the overall approach for measuring the performance of the VIPR reasoner.

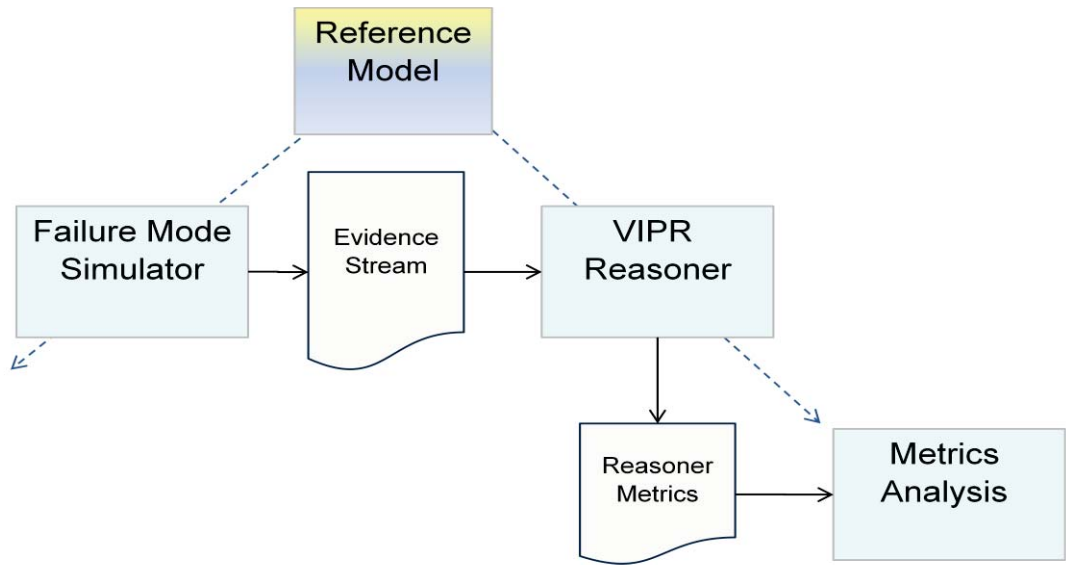


Figure 12: Components in the VIPR metrics analysis

For a given aircraft fault scenario, the failure mode simulator was used to generate an evidence stream that contained symptoms of the inserted faults as well as randomly generated evidence for spurious faults. Using the generated evidence stream, the VIPR reasoner was then executed for both the hierarchical and flat aircraft reference models. During its execution, the reasoner logged information about its execution that was analyzed by an offline tool that computed the accuracy, performance, and cost metrics reported in this section.

6.2 Metrics Generation Protocol

The hierarchical and flat aircraft reference models define more than 40 failure modes. Of these, 22 failure modes were considered complex (see Section 5.1).

We simulated single and multiple fault scenarios. The 22 single fault scenarios were each simulated 10 times using different evidence streams that each contained 0.1% randomly generated false evidence. Therefore the single fault data for each reference model was generated from the simulated insertion of 220 faults.

The 22 faults combine into 231 sets of double faults, and each two fault combination was simulated once. Therefore the multiple fault data was generated from the simulated insertion of 462 faults, two insertions for each of 231 test cases.

This means that the analysis was performed over data accumulated from running each reference model over 451 evidence streams that simulated 682 failure modes.

6.2.1 Failure Mode Simulator Parameters

We ran all failure mode test cases for a simulated time of 2,000 seconds, with evidence of faults provided in 10 second increments. Fault insertion, for both single and multiple fault scenarios, always occurred at simulated time 200, and randomly generated erroneous evidence could appear at any time during the 2,000 second simulation.

We assumed that the vehicle was able to completely process a set of evidence from LRU HM through the VHM before the arrival of the next set of evidence 10 seconds later. That is, we assumed that processing latency was always less than 10 seconds.

Each diagnostic monitor was supplied with false information 0.1% of the time. The aircraft reference models each contain about 40 diagnostic monitors, and during a simulation, each monitor is supplied with 200 values (2,000 seconds divided by the 10 second time increment). The generation of 8000 monitor values over a test case means that, on average, eight monitor values were caused by false information.

At simulation start, all monitors were considered inactive and became active at an exponential rate with a mean of 20 seconds. This means that by 20 seconds into the simulation, half of the monitors had been activated, and by the fault insertion time 200 seconds, there was a very high probability that all monitors were active.

6.2.2 Reasoner Parameters

The reasoner contains several parameters for tuning its operation. We focused on the DELTA_I parameter, which affects the reasoner's fault isolation sensitivity. With too high a value, the reasoner may have difficulty isolating the inserted faults, while too low a value may cause it to incorrectly isolate faults that were not inserted.

When the reasoner receives evidence of a failure, it computes the likelihood for each fault condition that could have caused the evidence. One criterion for achieving fault isolation is for the likelihood of the isolated fault to be much higher than the likelihood for other faults. DELTA_I is the measure for how much higher. For lower values, the reasoner requires less evidence to indict a failure condition. Consequently, the reasoner is faster at isolating failure modes but more likely to indict a failure condition that does not exist.

DELTA_I is expressed as the log of the ratios of the likelihood. Therefore, setting DELTA_I = 2 means that to achieve isolation, the fault's likelihood must be at least 100 times greater than the likelihoods of the other faults.

We tried three values of DELTA_I (1.5, 2.5 and 3.0) and determined that for these aircraft reference models and our analysis protocol, the 3.0 value produced the best results: fewest incorrect indictments without a significant drop-off in indicting the inserted faults.

6.3 Accuracy Analysis

The accuracy analysis was based on the VIPR reasoner's final state after each 2,000 second simulation.

As the reasoner receives evidence of failure modes, it builds ambiguity groups consisting of a group state and a set of failure modes. The likelihood that the particular failure condition is the cause of the evidence is associated with each failure mode. These likelihoods are relative to the other failure modes listed in the same ambiguity group. The likelihood of a failure condition in one ambiguity group cannot be compared to the likelihood of a failure condition in a different ambiguity group.

For analysis, ambiguity groups were in one of these two states at the end of the simulation:

Isolated	Isolated is a terminal state. In this state, the reasoner has completed analysis for the ambiguity group and has identified a failure mode, which is the only failure condition remaining in the ambiguity group. Additional evidence will not improve the solution.
Waiting	In this state, the reasoner has received sufficient evidence to form the ambiguity group containing one or more potential failure conditions, but the evidence is not sufficiently strong to indict any one of them. Typically, the likelihood of one of the faults is very high, but not high enough to meet the DELTA_I threshold.

In an operational environment, we expect the reasoner results to be presented to an aircraft technician trouble shooting the fault in three lists:

- The isolated faults
- The faults detected with high likelihood (but not isolated)
- Remaining fault conditions reported by reasoner (fault conditions listed in an ambiguity but with low likelihood)

We expect the technician to use this information to develop a troubleshooting strategy by subjectively comparing the value of isolated and detected faults and integrating this information with specific knowledge about the aircraft.

We expect the technician to focus on the isolated fault conditions first and use information about the other reported fault conditions only when either the isolated fault list is empty or when investigation of the isolated faults does not lead to the malfunction.

We used the following paradigm for computing the accuracy metric. If the reasoner isolated one or more faults, we computed accuracy using just the results for the isolated faults and ignored the results for faults that were detected but not isolated. However, if no faults were isolated, then we based the accuracy analysis on just the faults detected with high probability.

This procedure for measuring accuracy can understate accuracy for the multiple-fault scenarios. Consider a case in which the reasoner correctly isolates one fault (and does not isolate any other fault conditions) and correctly detects, with high probability, the second fault (and does not detect with high probability any other fault conditions). If the two faults were injected in two single-fault scenarios, the outcome for isolating the first fault would be "very good" and the outcome for detecting the second fault with high probability would be "good." However, if the two faults were injected during the same test case, the reasoner would have a "very good" outcome for isolating the first fault but a "poor" outcome for the second fault, since in the presence of an isolated fault, we would not consider that the faults were detected with high probability.

6.3.1 Accuracy of Flat and Hierarchical Models

We measured the reasoner accuracy to the flowing outcomes:

- Inserted fault conditions isolated
- Incorrect fault isolations
- Inserted fault conditions detected with high probability (but not isolated)
- Incorrect fault conditions detected with high probability
- Missed fault conditions: inserted fault conditions that were not even detected

We developed the following notation for marking these conditions:

I	Fault isolated.
D	Fault detected with high likelihood but not isolated.
*****	Inserted fault. For example, *I indicates that the inserted fault was isolated.
+	A fault that was not inserted. For example, I+ indicates a fault that was isolated but not inserted.
M	Missed fault condition.

The '*' and '+' annotations can both be combined with the 'I' and 'D' annotations. For example, a test outcome with the annotation '*I+' indicates that the inserted fault was correctly isolated, but other fault conditions were incorrectly isolated as well.

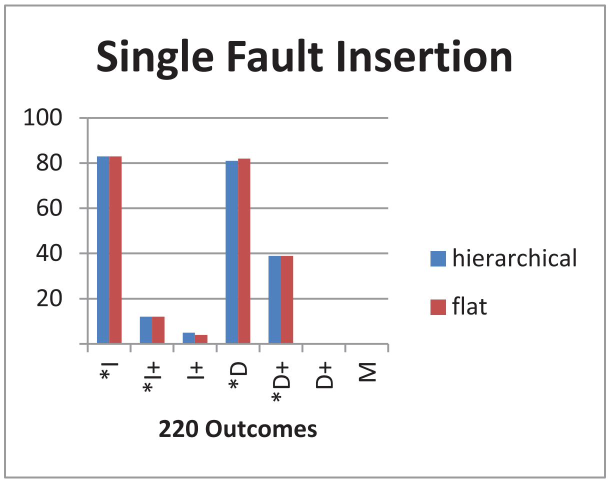
We ran the hierarchical and flat reference models for the same set of evidence streams and got very similar accuracy results for the two models, as shown in Figure 13 below.

Of the 220 single fault test cases, 38% ended with the best result: the inserted fault, and only the inserted fault, was isolated (the *I column in the figure). In an additional 5% of the test cases, the inserted fault was isolated, but other fault conditions were isolated as well. This outcome is less useful since, while it identifies the fault, it does so with ambiguity.

Considering test cases that contained no fault isolations, in 37% only the inserted fault was detected with high likelihood (the *D column) and in an additional 18%, the inserted faults were among several fault conditions detected with high likelihood.

In 2% of the test cases, the reasoner produced a result that would mislead the maintenance technician (the *I+* column) by not including the inserted fault among its list of isolated faults.

In summary, for the single fault case, the reasoner provided an unambiguous and correct result in 75% of the test cases and a correct but ambiguous result in an additional 23% of the test cases.



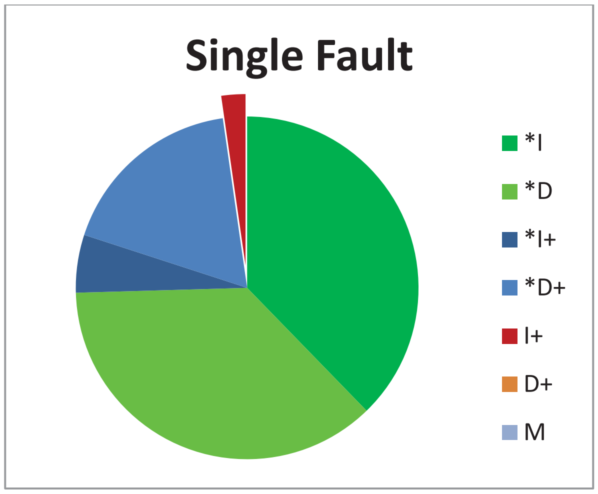
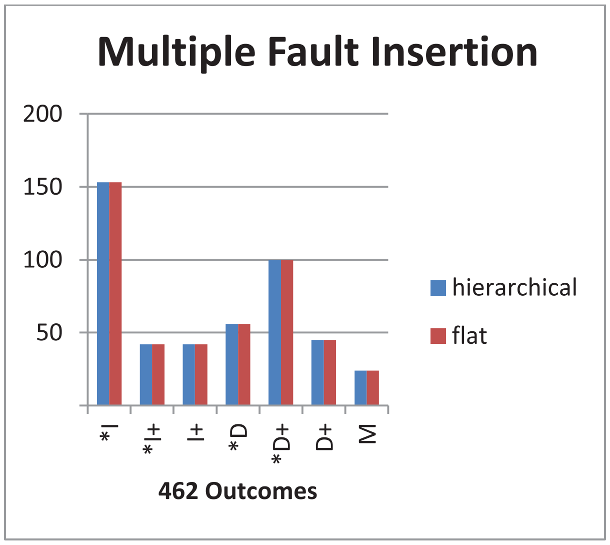


Figure 13: Results of single fault simulations

Outcomes for multiple fault case were not quite as good, as shown in Figure 14 below. The outcome was correct and unambiguous in 45% of the test cases (columns *I and *D) and correct but ambiguous in 31% of the test cases. In 14% of the test cases the reasoner indicted other fault conditions without indicting an inserted fault, and it failed to detect 5% of the fault insertions.



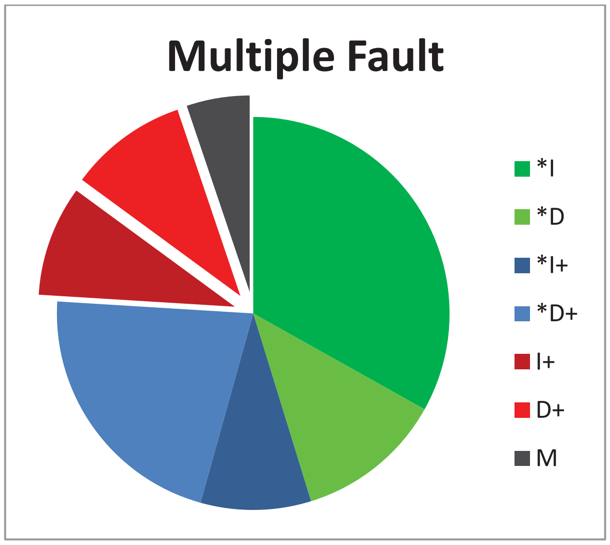


Figure 14: Results of multiple fault simulations

6.3.2 False Alarms

A false alarm is a fault detection caused by false evidence that appears in the evidence stream before fault insertion and that is not cleared by the reasoner by the end of the simulation. Twenty-one false alarms occurred in the 220 single fault test cases and no false alarms occurred during the 231 double fault test cases. Note that the 21 false alarms occurred in just three test cases; conversely no false alarms occurred in 448 of the 451 test cases.

6.4 Isolating Node

An important difference between the hierarchical and flat reference models is that reasoning is distributed in the hierarchical model (reasoning can occur at any of the LRU, area and vehicle levels), while, for the flat model, reasoning is centralized at the vehicle level. This difference is reflected in the table below, which reports the vehicle architectural level for each fault isolation. This data combines results for both single and multiple fault insertion, and includes all isolations, not just isolations of the inserted faults.

	Hierarchical	**Flat**
LRU	519	
Area	1	
Vehicle		519

While isolation for the flat reference model will always occur at the vehicle level, for this hierarchical model, isolation almost always occurred at the LRU level. This may be an advantage since computing resources at the LRU level tend to be less expensive and more available than at higher levels of the architecture.

6.5 Time to Isolate

We measure the time to isolate in steps of the discrete event simulation. Each step was 10 seconds long, and we assumed that computation for one step completed before the start of the next step. This assumption held true when computing latency was less than 10 seconds. So for example, if a fault required three steps to isolate, and a step was 10 seconds long and computing latency was less than 10 seconds, then the fault would take 30 seconds to isolate. Time to isolate was the same for the two reference models.

For the single fault test cases, the worst case time to isolate was 15 steps, and the average case was only 0.6 steps. The wide difference between the average and worst cases occurs because most isolations occurred during the step when the fault was inserted.

For the multiple fault test cases, isolation time was dramatically longer. The worst case time to isolate a fault was 153 steps and the average case was 13.6 steps.

6.6 Safety Impact and Accuracy of Prognostics

We tested two measures for prognostics. First, using the fault simulator, we tested the prognostic fusion accuracy. This measure tests the accuracy of the prognostics fusion rule. The fusion rule was found to be accurately implemented (example in Figure 15).

Our second prognostic measure was to generate monitors to detect precursors to significant safety incidents such as in-flight engine shutdown. Figure 16 shows three such incidents. In the first incident, starting from the left of Figure 16, the precursors were detected onboard and isolated approximately 30 flights early, therefore, VIPR could provide the maintainer time to intervene and prevent the safety incident. While the "discovered prognostic monitor" is very accurate, VIPR takes an "engine-wide view" and also suspects some secondary damage in the hot section, as defined in the manufacturer's FMEA.

In the second case, the precursors were detected and isolated approximately 20 flights early. While the "discovered prognostic monitor" is very accurate, VIPR is taking an "engine-wide view" and looking for more supporting evidence such as those defined in the manufacturer's FMEA.

VIPR conclusions from the third case show that the precursors were detected onboard with high likelihood. The precursors appear as different problems in the two engines. VIPR uses cascade reasoning and active query of the remaining two engines (#2 and #3) to identify a common cause – fuel delivery manifold. This then suppresses the net result of a high false alarm monitor at individual engine level.

These three cases show the prognostic accuracy of VIPR case by case and have also illustrated how the VIPR approach detects precursors to safety incidents well in advance of the actual event (in-flight shutdown).

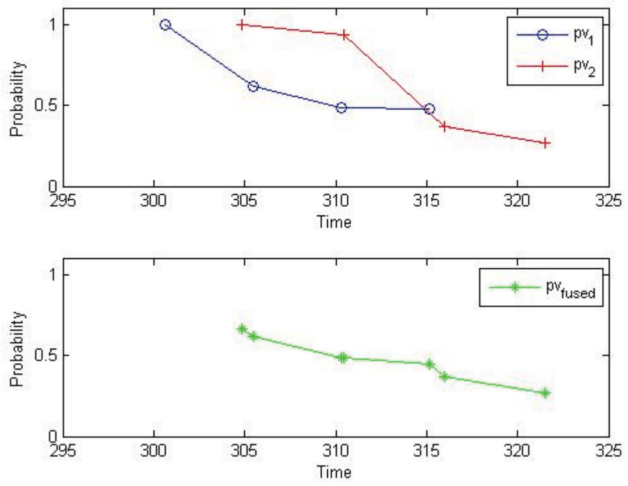


Figure 15: Fusion of two prognostic vectors

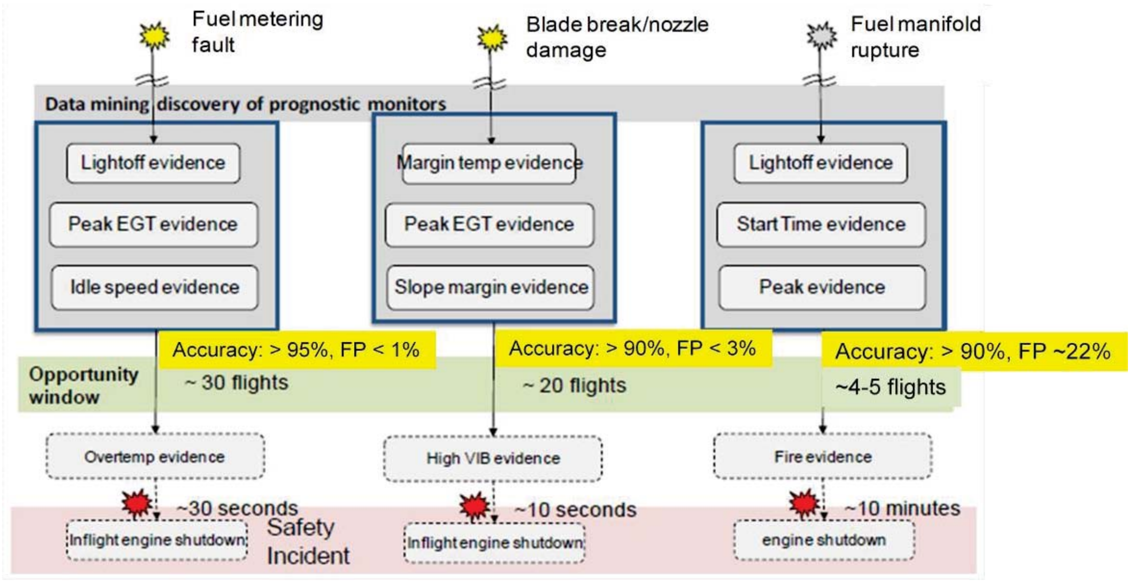


Figure 16: Discovery of prognostic monitors using data mining on an airlines database

VIPR prognostics do not predict the time when a failure will occur or otherwise provide an estimate of the remaining useful life of a component.

6.7 Communications Volume

Communications volume is characterized by the number of messages sent by the reasoner, and the size of the messages. Message format was defined by the ARINC 624 standard. Tabulations for the two reference models under single and multiple fault scenarios are shown in Table 1.

As expected, the distributed hierarchical model required more messages than the flat model to achieve equivalent fault detection and isolation results. For the single and double fault test cases, the hierarchical model required 28% and 29% more messages, respectively, than were required for the flat reference model. The messages for the hierarchical model tended to be a bit larger as well, and in total, the hierarchical model required the transmission of about 40% more bytes.

The multiple fault test cases required significantly more messages than the single fault cases. On a per-inserted-fault basis, the double fault test cases sent 70% more messages per fault than the single fault cases, and twice as many bytes. This held true for both the hierarchical and flat models.

Table 1: Communications volume for the two reference models under single and multiple fault conditions

	Transactions	Bytes	Bytes/Tran	Trans/Fault	Bytes/Fault
Hierarchical, single fault	40,972	7,907,861	193	178	34,382
Flat, single fault	32,053	5,594,980	175	139	24,326
Hier, multiple fault	142,304	31,805,968	224	308	68,844
Flat, multiple fault	110,264	23,019,124	209	239	49,825

6.8 Communications Latency

Safety-critical aircraft communications systems, such as AFDX and ASCB, are designed with statically defined periodic schedules. Therefore, each message that can be sent during operational use of the aircraft must fit into a preallocated slot in the schedule for that message. This paradigm works well for systems that continuously produce information that must be propagated to other parts of the aircraft.

In contrast to the periodic communications system, reasoner messaging is sporadic. The reasoner does not send messages when there is no evidence of faults or the fault evidence is unchanging. However, the arrival of evidence triggers activity that results in the transmission of a sequence of messages.

6.8.1 Latency Model

Computational latency is the time required to execute a transaction. In an avionics system, we expect the communication latency to be much longer than the processing latency, and hence, considered just the communication's contribution to latency.

The reasoners produce transactions at sporadic times. That is, when the monitors are not reporting any fault symptoms, the reasoners are quiet. However, when a fault occurs, one or more monitors may initiate transactions, the effect of which can then ripple through the network of reasoners.

Avionics communications protocols tend to have periodic schedules that are static, and hence are not well-suited for handling sporadic communication. In a static, periodic schedule, every message that can be sent must be allocated a place in the predetermined schedule. If the message is not periodic and needs to be sent infrequently, then the message could be allocated a very small part of the communications bandwidth. But when there is a flurry of activity, a small bandwidth can result in long latencies. Alternatively, the message could be allocated a larger bandwidth, but at a loss in communications utilization since the larger bandwidth would be used infrequently.

This effect is illustrated in Figure 17, which shows the arrival of three messages, represented by the yellow spikes, and message transmission bursts for two communications rates, shown in green and blue. Message transmission begins on the arrival of the first message and continues through the second message, because the transmission of the first is not completed when the second arrives—at either rate. The green rate is high enough to complete sending the second message before the arrival of the third, while the blue rate is not. Hence, when the third message is ready to be sent, the green rate begins sending it immediately, while the blue adds it to its transmission queue.

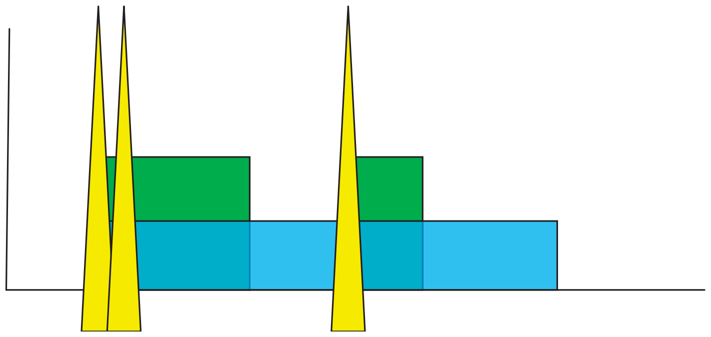


Figure 17: A notional representation for latency when sending a burst of messages over a periodic communications system.

We express communications latency by extrapolating from the durations of the message transmission bursts. In Figure 17, the green rate requires two bursts to transmit the three messages while the blue rate has just the one long burst.

We measured both the average and maximum durations of bursts for a range of transmission rates and fault conditions and for the two reference models. While average burst length tends to be very low, the more important metric is the worst case burst, which can be quite long.

For each fault scenario, the maximum burst was calculated for bandwidths in the range 10 bytes/second, 100 bytes/second, 1000 bytes/second, etc., stopping when the maximum burst for a fault scenario was less than ten seconds long. The fault scenarios required a message transmission rate of 10,000 bytes/second to achieve the worst case ten second threshold.

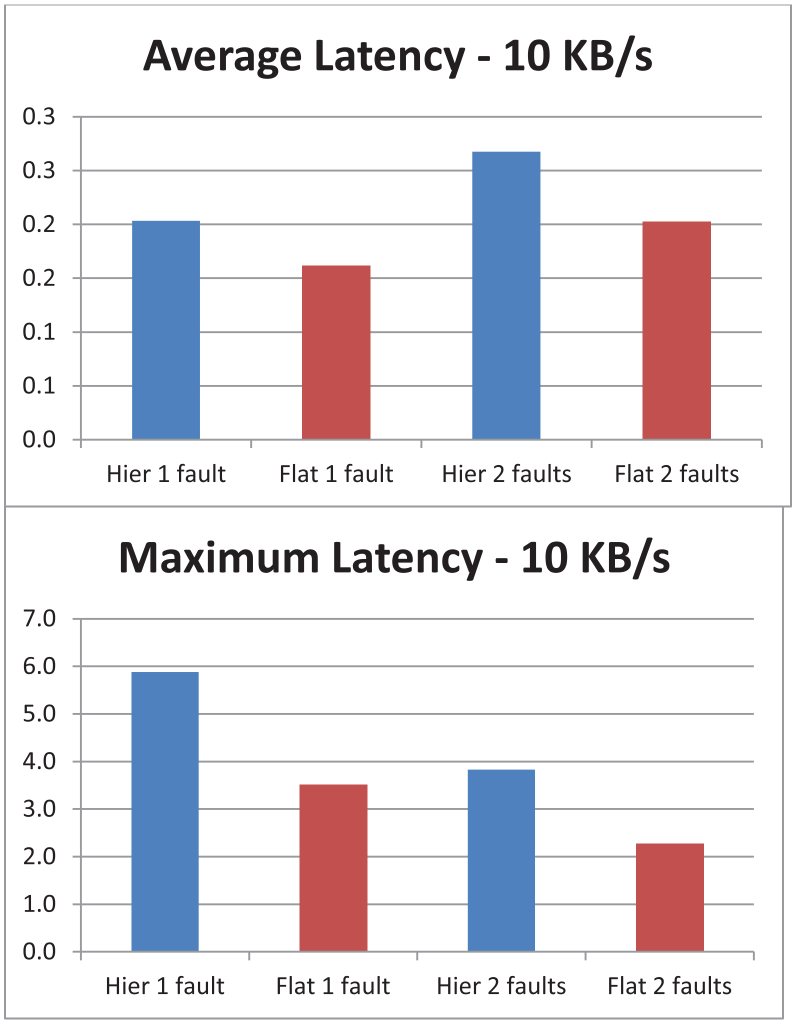


Figure 18: Communications latency computed from the single and multiple fault simulations

6.9 Cost Analysis

6.9.1 Cost Model

The VIPR reasoner consists of a collection of entities that exchange messages. Messages tend to flow from the sensor monitors and up through the hierarchy of LRU, area and vehicle reasoner entities, and finally to a consumer of the vehicle health information, such as an aircraft display. This flow of information is depicted in Figure 19.

Cost is computed as the sum of the costs for each reasoner transaction. A transaction embodies the computation and communications required to send a message from a source entity to a destination entity, as depicted in Figure 20.

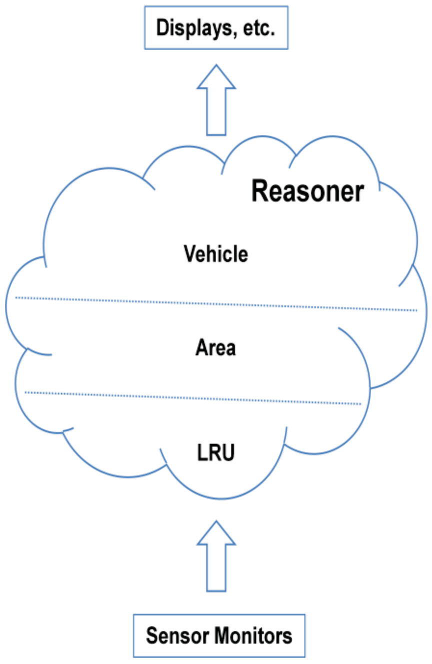


Figure 19: Fault information flows from the sensor monitors through the reasoner entities to the users of the reasoner's conclusions

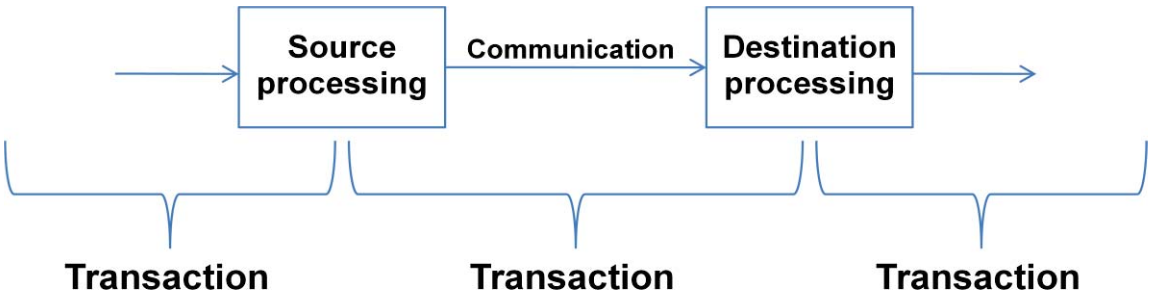


Figure 20: Cost is computed as the sum of transaction costs.

A transaction includes half of the source entity’s processing, the communication needed to transmit the message and half of the destination’s processing.

The model assumes that an entity’s processing cost is the same for all transactions, and that communications cost is proportional to the size of the message being transmitted. Each reasoner entity is assigned these two cost parameters:

1. P expresses the entity’s processing cost.
2. C expresses the entity’s communications cost for transmitting/receiving a byte of information.

The communications overhead for a transaction is assumed to be shared equally by source and destination, hence the cost of a transaction for a message containing M bytes is:

$$TransactionCost = \frac{Psource + M * Csource}{2} + \frac{Pdestination + M * Cdestination}{2}$$

The ratio of P:C specifies the processing cost relative to the per byte communications cost. This ratio depends on a specific vehicle architecture. To gauge the cost over a range of values, we performed the cost analysis using ratios of 10, 100, 1000, and 10,000 to one (that is, the processing cost is equivalent to the cost of sending 10 bytes, 100 bytes, 1,000 bytes or 10,000 bytes, respectively).

We assigned all entities at the same architectural level (LRU, Area, Vehicle and Display) the same values of P and C. We expected the cost of processing resources to be specific to the architecture of each vehicle, so we performed the analysis for these three cases:

1. Cost does not vary by level	(LRU = 1, Area = 1, Vehicle = 1, Display = 1)
2. Cost increases linearly by level	(LRU = 1, Area = 2, Vehicle = 3, Display = 4)
3. Cost increases exponentially by level	(LRU = 1, Area = 2, Vehicle = 4, Display = 8)

For a given entity, the value of its C parameter is the value for the level that contains the entity. The value of its P parameter is the P:C ratio selected for the simulation multiplied by the level value. For example, if cost is assumed to be linear across levels and the selected P:C ratio is 1,000, then the cost parameters for a reasoner entity at the area level are:

1. C = 2
2. P = 2,000

The cost for a transaction containing 200 bytes that is sent from an entity in the area level to an entity in the Display level is the average of the costs at the two levels. The cost expression above evaluates to 3,600 for P:C = 1,000 and a linear increase in cost with level:

$$TransactionCost = \frac{2{,}000 + 200 * 2}{2} + \frac{4{,}000 + 200 * 4}{2}$$

Reasoner entities sometimes send messages to themselves. In this case, the communications cost is assumed to be zero, and we computed the processing cost entirely from entity's processing cost metric.

6.9.2 Cost Analysis Results

We combined the cost data for the 220 single fault insertions and 231 multiple fault insertions and compared the relative costs for the hierarchical and flat reference models. The cost advantage for the flat model is that it requires less communication than for the hierarchical model. The advantage for the hierarchical model is that it can have entities on any of the levels while all entities in the flat model exist at the vehicle level, which may be a more expensive computing resource.

The charts in Figure 21 show the relative cost of the hierarchical model to the flat model over the four combinations of the P:C ratio and the constant, linear and exponential cost assumptions.

Costs are shown relative to the lowest cost option, the flat model with P:C ration = 10, and constant cost across levels of the architecture. Note that this cost value is the missing red column in the first chart—the value of the column is one, but when displayed on a logarithmic axis that starts with value 1, the length of the column is zero.

As expected, the flat model is consistently cheaper when computing cost is constant across the system architecture because it requires fewer transactions. The hierarchical model is less costly for systems where computing costs are higher at the higher architectural levels.

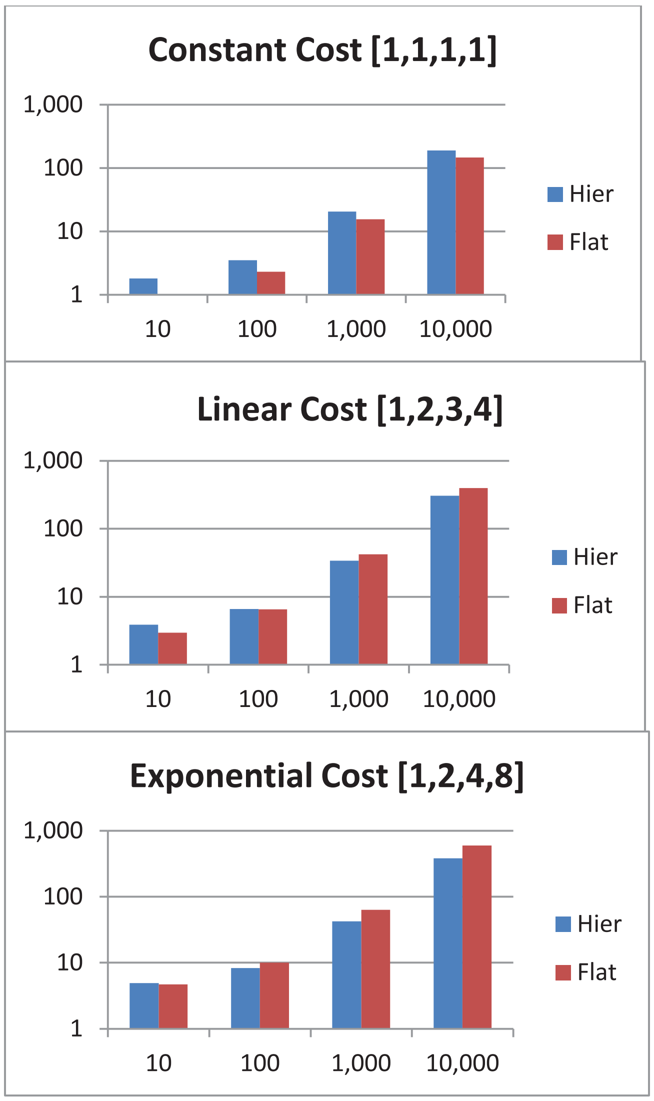


Figure 21: Relative cost of hierarchical and flat reference models

For the P:C = 1000 case, the flat model is 32% less costly than the hierarchical model when costs are constant across architectural levels, but the hierarchical model is 24% less costly for a linear increase across levels and 51% less costly for an exponential increase across levels.

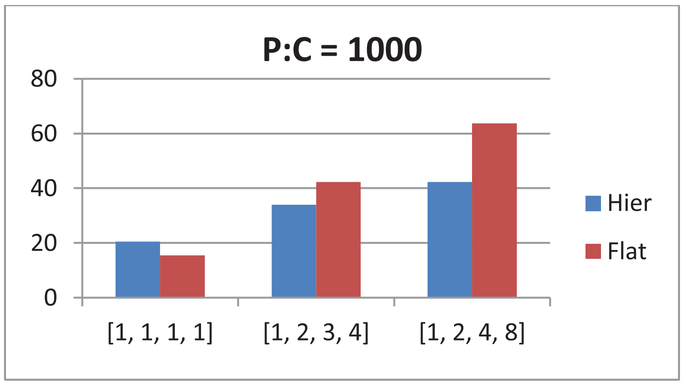


Figure 22: Relative costs for the two models, assuming processing cost is equivalent to the communications cost for a 1000 byte message

6.10 Prognostic: Time to Failure Metrics

A measure of value of prognostic warning is how far into the future it can predict a fault. We studied the effect of these three monitors reported in the airline database for predicting the engine bleed fault:

1. Rising start
2. Fast start
3. Fuel HMA

The first two monitors are trend monitors; they predict the occurrence of a fault by tracking the changes in a parameter value over time. The third is a "super" diagnostic monitor that integrates several parameter values to predict a looming failure. Unlike a typical diagnostic monitor, this monitor fires before the actual fault occurs, and hence is a prognosticator. Unlike trend monitors which use the change in a parameter's value to predict a fault, a "super" monitor bases its prediction on the current values of several parameters.

Figure 23 below shows the fault prediction based on the rising start trend monitor.

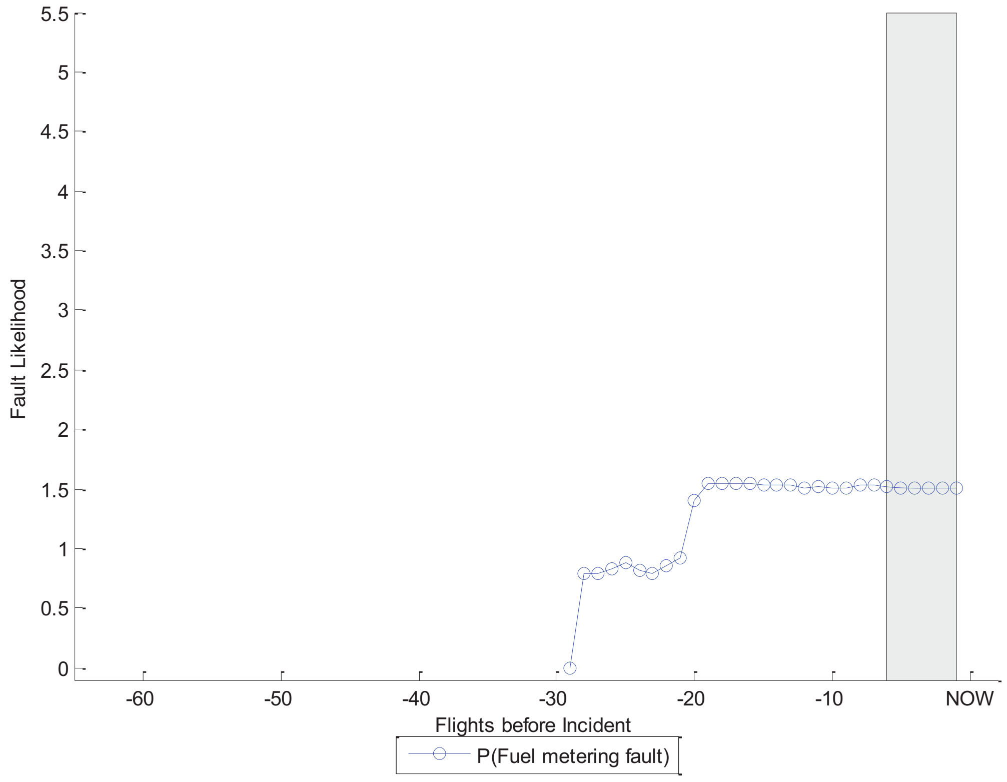


Figure 23: Fault prediction using the Rising Start trend monitor

The Y-axis in this figure indicates the relative likelihood of a fault occurrence against the likelihood that the fault will not occur. It is plotted on a log scale, so the 0 point indicates that there is an equal probability that the fault will occur or not.

The trend monitor first started predicting a fault about 30 flights before the actual fault occurrence. One flight later the fault:no fault ratio increased to 10:1, and 20 flights before the fault occurrence the likelihood increased to 13:1.

The Fast Start trend monitor yielded a similarly shaped prediction, as shown in Figure 24.

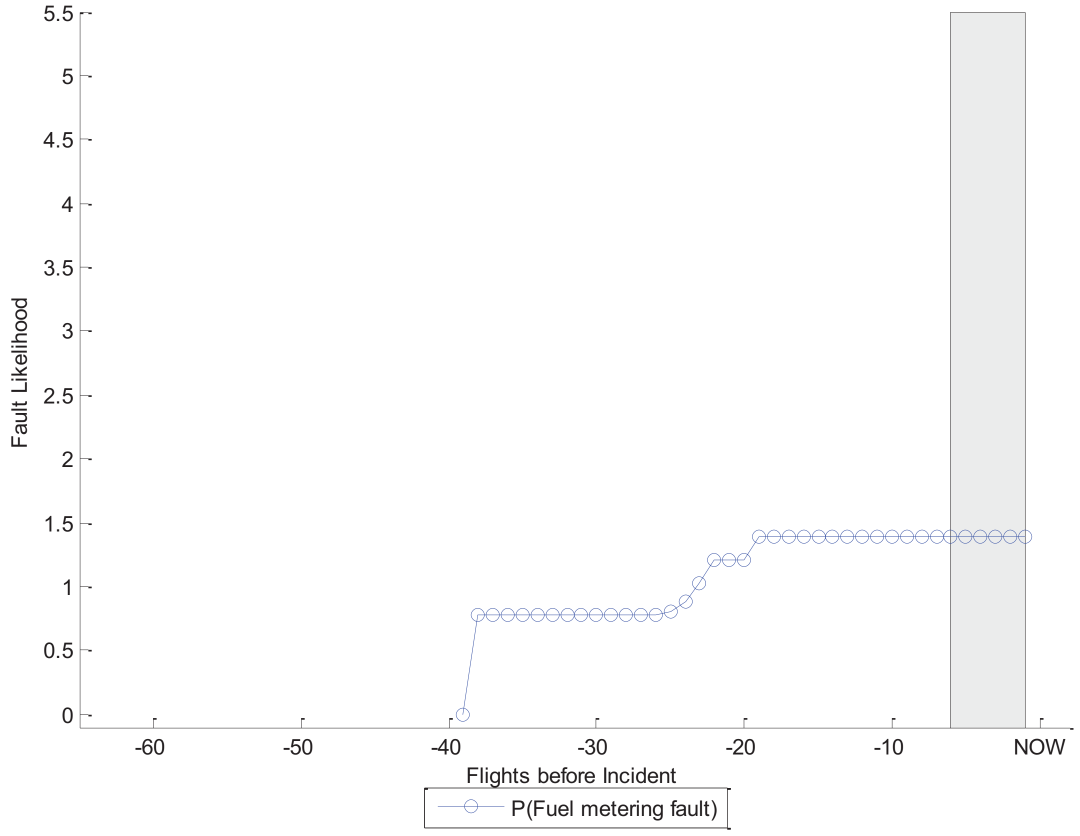


Figure 24: Fault prediction using the Fast Start trend monitor

For this monitor, the prediction of a fault started about 10 flights earlier but yielded a somewhat lower likelihood than did the Rising Start trend monitor.

Fusing the results of these two trend monitors improved the prediction by incorporating the earlier detection of the Fast Start monitor but also retained the lower likelihood that monitor. The result f fusing the two trend monitors is shown in Figure 25.

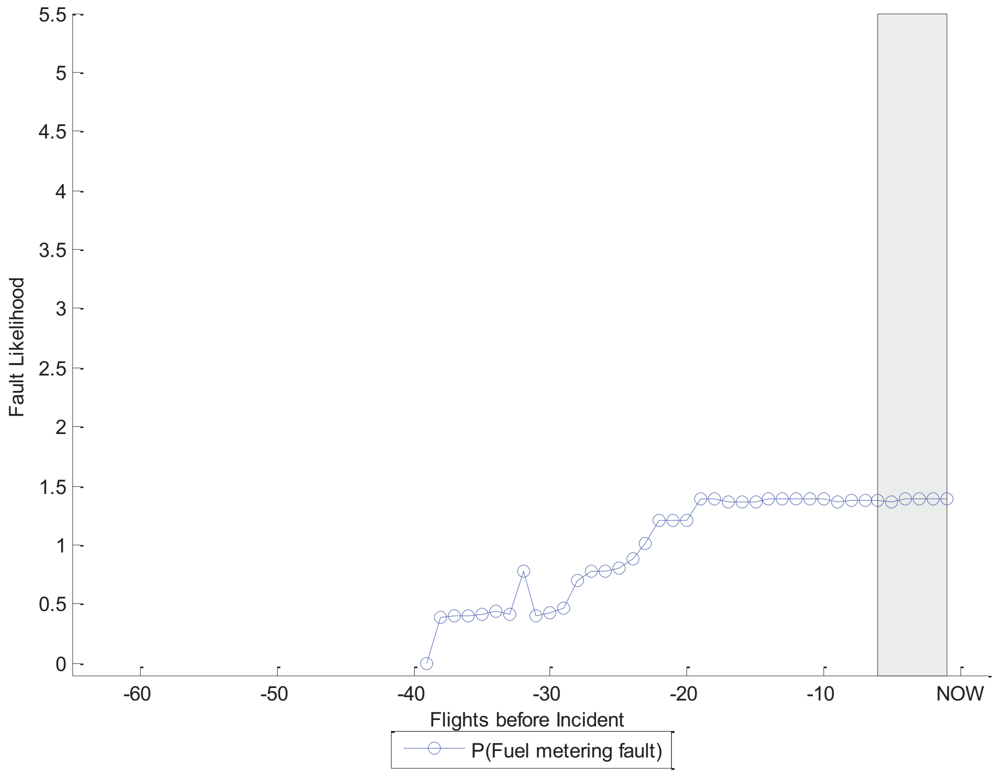


Figure 25: Fault prediction from fusing results from the Rising Start and Fast Start trend monitors

The shape of the prediction curve for the Fuel HMA diagnostic monitor is different from the shape for the trend monitors, as shown in Figure 26. A diagnostic monitor fires when its preconditions are met, and in this case, the preconditions were met, and continued to be met for 27 flights before the failure.

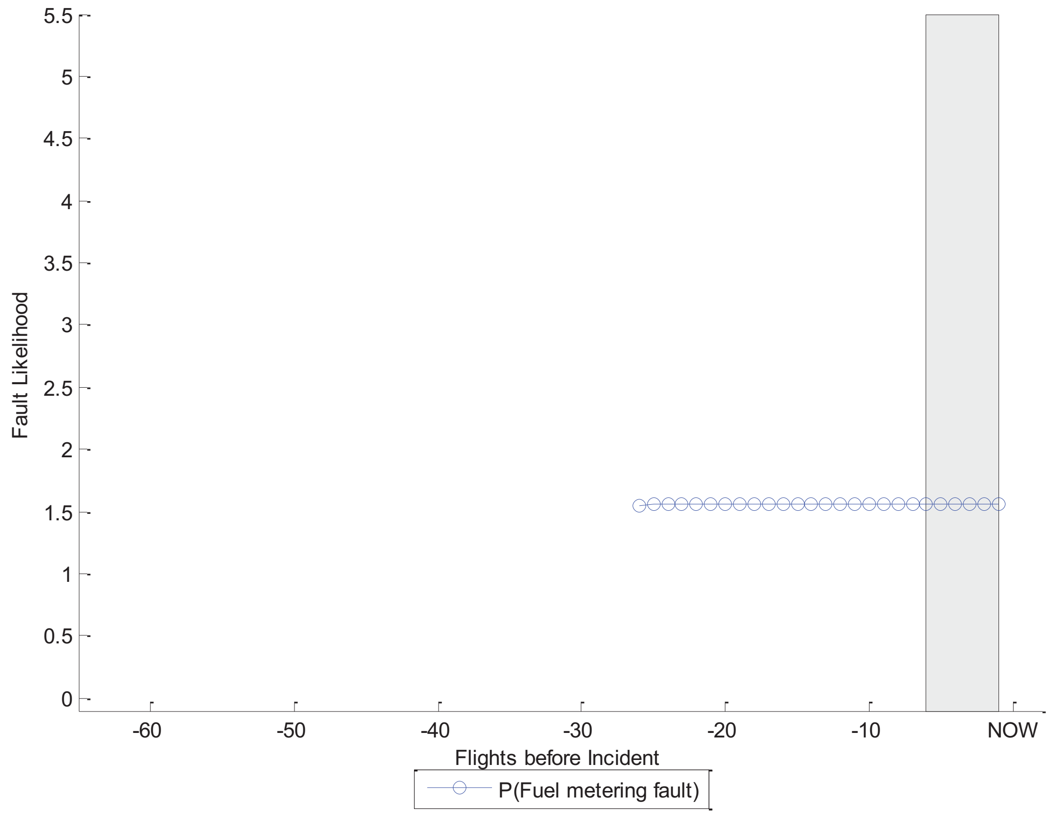


Figure 26: Fault prediction using the Fuel HMA diagnostic monitor

The best results are achieved by fusing the results of all three monitors, as shown in Figure 27. The fused results begin predicting the fault nearly 40 flights before the occurrence and with a likelihood of greater than 2:1. By 25 flights before fault occurrence, the likelihood has increased to 10,000:1 and increases to 100,000:1 by 20 flights before the fault occurrence.

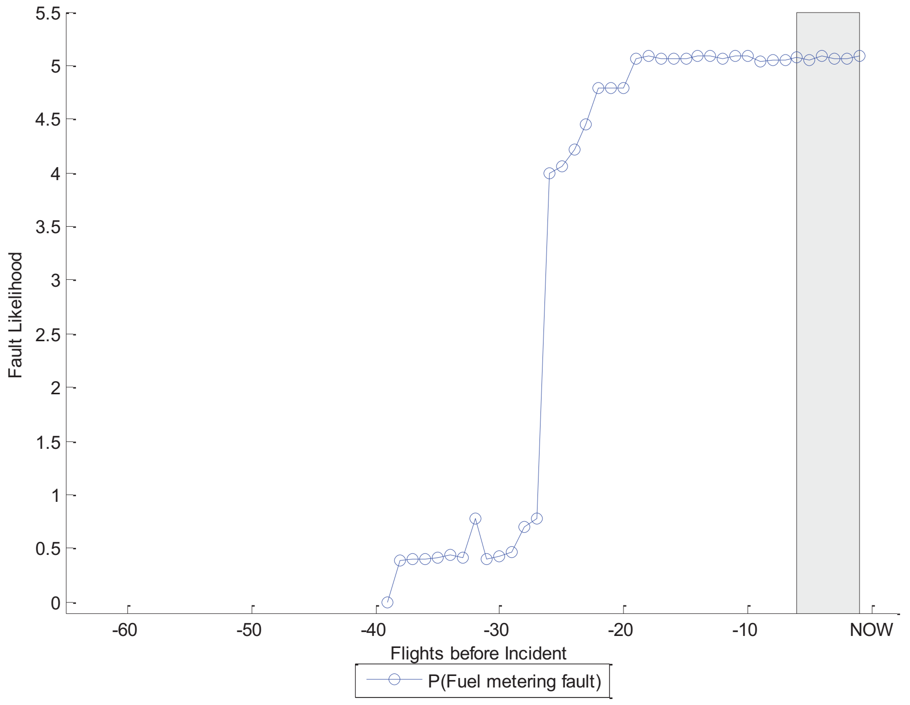


Figure 27: Fault prediction fusing all three monitors

Another benefit of the fused result is that it yields fewer false alarms. The Fuel HMA monitor on its own is prone to generating false alarms. But the influence of the other two monitors, which do not by themselves generate many false alarms, greatly reduces the incidence of false alarms. So the prediction from the fused results provides the longest forecast before incident, the highest confidence level, and a low false alarm rate.

6.11 Reasoner Floating point Operations

The number of floating point operations (FLOP) is calculated as follows:

1. The calculation in VIPR is event-driven and consists of several steps.
2. Two events T_1, T_2 drive the calculations:
    a. T_1: A member system provides it a new diagnostic or prognostic monitor.
    b. T_2: The active query provides it receives a parametric value from any of the aircraft member system.

Only five steps within VIPR do floating point operations. Other operations are a comparison step, array indexing or messaging; these are excluded in the FLOPS calculation. These steps are labeled S_1, S_2, S_3, S_4, S_5 as follows:

1. S_1: Prognostic Monitor Generation. This step occurs *only* when the calculations are driven by T_1. Since VIPR supports four mechanisms, this step is classified as follows:
    a. Condition indicator-based – options include linear and hidden-state trending.
    b. Exceedance-based – options include simple counting and latched-counting.

The number of floating point operations is a function of the buffer size (the number of samples in the trend history window) and the prediction size (the number of samples in the prediction window). Since both these include the calculation of statistical standard deviation, the FLOP count is a function of $W^2 + B^2$ where W is the prediction window size and $B$is the buffer size. Typically, the number of points in the buffer window and the trend window will be the same order-of-magnitude. We approximate this as$O(B^2)$. Further, all evidence defined in the reference model may have a prognostic monitor. In this case, these operations will be repeated $O(E)$ times, where $E$is the number of evidence defined in the VIPR reference model. Hence, the FLOP count for this step is $O(EB^2)$.

2. S_2: Prognostic vector fusion. The fusion process includes linear interpolation for lining up two vectors and obtaining a mean at each piecewise-interpolated point. Since an interpolation function consumes $O(B)$ FLOPs, we use the same notation B to indicate the average number of points in the trend window and, hence, the samples in the prognostic window. Further, the fusion operation is distributive, which makes the fusion independent of the order in which two vectors are fused. In the worst case, if the reference model defines Eevidence, we need $(E-1)$ fusion at each step. Hence, the FLOP count for this step is $O(EB)$.
3. S_3: Hypothesis likelihood update. This uses a noisy-OR Bayesian model to calculate the posteriori probabilities for various fault condition hypothesis. Specifically VIPR calculates the log-likelihood values for various fault condition hypothesis. If the reference model defines Eevidence, in the worst case, a given failure mode can be connected to every one of these evidence. Further, evidence has a detection probability and a false alarm probability or 3 floating point calculations per evidence connected to a failure mode. Hence, the FLOP count for the noisy-OR calculation per failure mode hypothesis is $O_{log}(3E)$; here, the subscript log indicates that the number must be multiplied by the FLOP required to perform a natural logarithmic function calculation.

   VIPR performs the noisy-OR calculation for both single and two-fault hypothesis. In the worst case, all F failure modes defined in the reference model may be occurring simultaneously. In this case, we will have $\frac{F(F+1)}{2}$ hypothesis for which the noisy-OR calculations must happen, which implies we will end up with $O_{log}(F^2E)$ floating point operations for this step.
4. S_4: Likelihood normalization. Normalizing is done only for single fault hypothesis. It involves subtracting a minimum value and dividing the result by a maximum value—the three-FLOP-per-single-fault hypothesis. Hence, this step will need $O(F)$ FLOPs.
5. S_5: FM distance. New failure modes are assigned to the ambiguity group of a fault condition using a pairwise distance calculation. Since the number of F failure modes are pre-defined in the reference model, there can be only $\frac{F(F+1)}{2}$pairs; hence, this step will need $O(F^2)$ FLOPs every time a change is made to the reference model and not every VPR update cycle.

Summarizing from the above calculations, we conclude that the number of FLOP per VIPR update step is bounded by an upper limit. The order of magnitude of this upper limit is:

$$VIPR\ flop \leq O_{log}(F^2E) + O(EB^2)$$

Here:

1. F is the number of failure modes defined in the reference model, E is the number of evidences defined, B is the number of samples used for generating prognostic monitors. Further, we assume that the prediction window for VIPR will be $O(B)$.
2. O_{log} denotes the order of magnitude for performing a single precision logarithmic calculation. The exact number of steps will depend on the processor architecture.

The run-time computations of VIPR described above are summarized in Table 2.

Table 2: Floating point operations within VIPR and its upper bound.

Assumptions	**FLOPs**	**Notes**
S_1: CI-based prognostic	$O(EB^2)$	Per VIPR update cycle
S_2: Prognostic vector fusion	$O(EW)$	Per VIPR update cycle
S_3: Hypothesis likelihood update	$O_{log}(F^2E)$	Per VIPR update cycle
S_4: Likelihood normalization	$O_{log}(F^2E)$	Per VIPR update cycle
S_5: FM distance	$O(F^2)$	Per new reference model load
Total	$\leq \boldsymbol{O_{log}\left(F^2E\right)} + \boldsymbol{O(EB^2)}$	Upper bound

6.12 Other Metrics

The VIPR software contains about 7000 lines of MATLAB code, half of which implements the reasoner. The remaining code is split among messaging, monitors, parsing the reference model (about 1000 lines each), and the fault simulator (500 lines). For a reference model that defines N failure modes, the reasoner will allocate data structures that consume O(N^2) storage space.

Other software metrics, such as code complexity, links between software components, etc., have value when measured for a production grade implementation and were not computed for the prototype VIPR software.

7 Metrics Derived from the VIPR Hardware-in-the-Loop Demonstration

The VIPR program included a hardware-in-the-loop (HIL) demonstration (Figure 28) that featured Honeywell's LaserRef VI inertial reference unit (IRU). The demonstration configuration allowed us to expand the metrics analysis to include data from commercial avionics equipment.

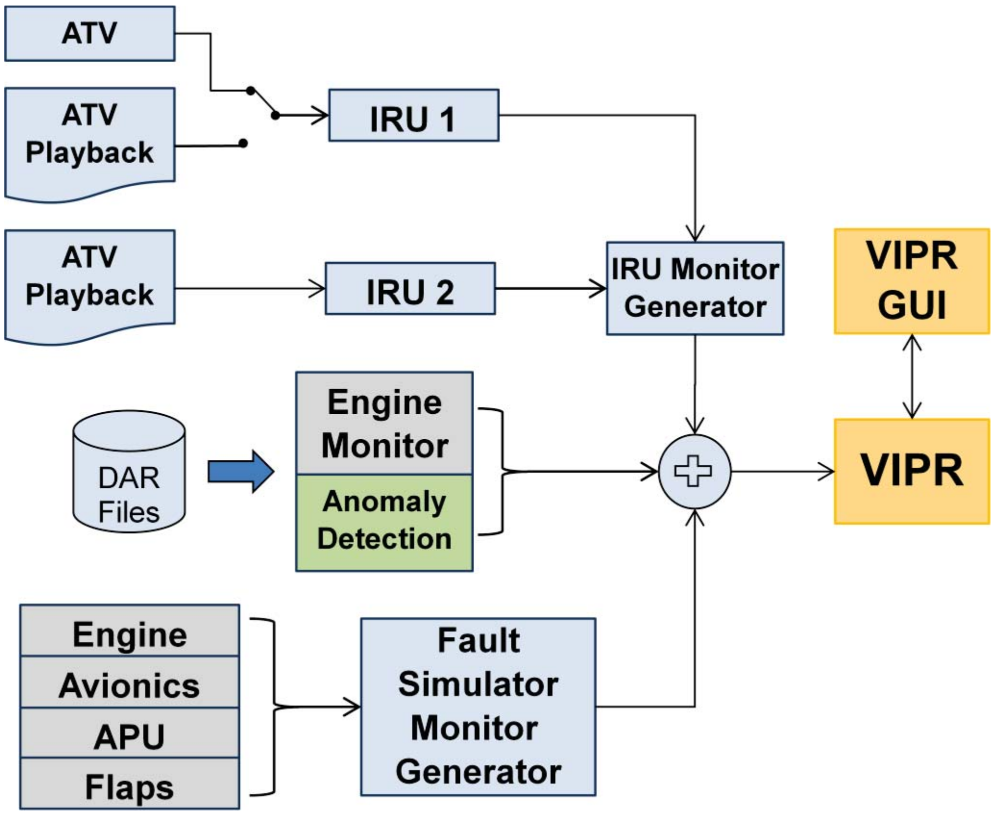


Figure 28: The logical architecture of the VPR Hardware-in-the-Loop demonstration.

A version of the LaserRef VI IRU, known as the acceptance test vehicle (ATV), was used in the demo. An ATV is a LaserRef VI without sensors for measuring linear and angular acceleration. An ATV is used in place of a LaserRef VI during product development when providing test sensor data from a file is more desirable than using the actual sensors.

The demo consisted of providing the VIPR reasoner with fault evidence from the fault simulator, the airline database, and the ATV. An evidence stream from the ATV could be received either from the ATV in real-time or replayed from a file of recorded ATV output. The demo configuration contained evidence streams for two IRUs, one of which had to be the replay of a recorded ATV output file and the other could either be the ATV producing the evidence in real-time or a second replay.

The LaserRef VI output consists of about 30 parameters for expressing the navigation solution and device status. The output is formatted as ARINC 429 words and transmitted at a 50 Hz rate. For the demo, we replaced the ARINC 429 interface with Ethernet and used it to send a data structure containing about half of the LaserRef VI outputs. Because the PC on the receiving end could not keep up with the LaserRef VI 50 Hz output rate, we reduced the transmission rate to 25 Hz.

ATV output processing occurred in an output process that gathered the data, packed the data into a structure, and sent the data to a TCP protocol stack for transmission to the PC. Using a development tool provided for the ATV, we measured the CPU time of the output process and incremental overhead on the TCP network processes for sending our data structure at about 4% of the processor's capacity. This value is consistent with the performance of the LaserRef VI, which for production usage is allocated

17% of the CPU throughput. This larger budget must support output at 50 Hz instead of 25 Hz, and the full complement of LaserRef VI outputs, while in our configuration we transmitted only half. However, since the output used in the demo is already being sent for use by other aircraft systems (the FMS in particular), VIPR would pose no additional processing on the LaserRef VI.

The evidence stream generated by the ATV had high fidelity and did not compromise the reasoner's accuracy. Although the simulation environment used for gathering metrics data did not include the ATV, evidence received from the ATV after processing by the diagnostic monitors appears to the reasoner the same as monitor input from evidence generated by the fault simulator. Hence, adding the ATV to the simulation environment would not have affected results, but would have greatly complicated the process of gathering the metrics because of the difficulty of performing Monte Carlo simulations using the ATV. In addition, the ATV's essential need to run in real-time would have significantly slowed the data gathering, since simulations that don't include the ATV can run much faster than real-time.

8 Summary and Conclusions

We have computed metrics that measure the Reasoner's accuracy, latency, communications bandwidth, computational cost and the rate of false alarms. Data was gathered from the insertion of 22 complex faults in both single and multiple fault cases, and for both hierarchical and flat aircraft reference models.

For each single fault condition, we generated 10 evidence streams for the inserted fault that also contained 0.1% erroneous fault evidence and then ran simulations for each evidence stream. For each of the 231 two fault insertion cases, we generated a single evidence stream containing 0.1% erroneous evidence and ran simulations for each of these evidence streams. A total of 902 simulations containing 1364 fault conditions were run.

From the simulation results, we conclude the following:

- Accuracy — The reasoner's ability to correctly isolate a failure from a given evidence stream is dependent on the quality of the evidence stream and the correctness of the reference model. When simulated with evidence streams that contained 0.1% erroneous data, the reasoner correctly and exactly identified the inserted faults 75% of the single fault cases and 45% of the multiple fault case. In addition, it correctly identified the inserted faults, but not uniquely, for an additional 23% of the single fault insertions and 31% of the multiple fault cases. Therefore, the Reasoner correctly identified 98% of the single fault insertions and 76% of the multiple fault insertions.

  Accuracy for the hierarchical and flat reference models were the same.

  The prognostics were accurate on three cases discovered from the airline database. The case studies on prognostics precursors have shown that the VIPR approach detects precursors to safety incidents multiple flights in advance of the actual event (in-flight shutdown).

- Isolation time — For the single fault test cases, the typical time to isolate was immediately after fault insertion. The worst case time to isolate was 15 steps and the average time was 0.6 steps.

The time to isolate for the multiple fault scenarios was about 10 times longer than for the single fault insertions: the worst case time to isolate was 153 steps and the average case was 13.6 steps.

- Communications bandwidth — For sending the ARINC 624 messages generated by the reasoner over a periodic safety critical communications system, we computed that 1 KB/second bandwidth would yield, on average, message latency of 1-3 seconds depending on reference model and number of faults inserted. However, to reduce the worst case latency below the 10 second goal for all simulations required a 10 KB/second communications bandwidth.
- Isolating node — For the flat model, the isolating reasoner entity is always in the vehicle node. However, for a hierarchical model, reasoning may occur at any node. For our aircraft hierarchical model, isolation occurred at the LRU level 519 times, once at the Area level, and never at the Vehicle level.
- Communications volume — The flat model required 22% fewer messages and 28% fewer bytes in total than did the hierarchical model.
- Computation cost — Where computation cost is the same at all aircraft levels (LRU, area, and vehicle), the computation cost for the flat model was lower than for the hierarchical model because the flat model requires fewer transactions to achieve the same results as the hierarchical mode.

  However, where computing at higher nodes is more expensive than at the lower levels, the distributed hierarchical model is less costly because it can perform much of its computation on the lower cost computing resources.

  The cost of computation for the reasoner is proportional to the log of the number of faults square and the square of the number of samples used for generating prognostic monitors.
- False alarms — The 0.1% rate of false evidence generated false alarms in only three of the 902 simulations run.
- ATV Integration — The high quality of the LaserRef VI self-diagnostics allowed us to use the device's existing output for input to the VIPR software; consequently, integrating the LaserRef VI with VIPR added no overhead to the device's operation.

9 References

1. C.S. Byington, M. Watson, P Kalgren, and R. Safa-Bakhsh, “Metrics Evaluation and Tool Development for Health and Usage Monitoring System Technology,” *Third International Conference on Health and Usage Monitoring - HUMS2003*, Melbourne, Australia, G.F Forsyth (editor), pp. 27-35, 2003.
2. T. Kurtoglu, O. Mengshoel, and S. Pol, “A Framework for Systematic Benchmarking of Monitoring and Diagnostic Systems,” *International Conference on Prognostics and Health Management Conference*, Denver, CO, 2008.
3. A. Saxena, J. Celaya, E. Balaban, K. Goebel, B. Saha, S. Saha, and M. Schwabacher, “Metrics for Evaluating Performance of Prognostic Techniques,” in *International Conference on Prognostics and Health Management (PHM08)*, Denver CO, 2008.
4. T. Felke, G. Hadden, D. Miller, and D. Mylaraswamy, “Architectures for Integrated Vehicle Health Management,” *AIAA*-2010-3433, 2010.
5. G. D. Hadden, D. Mylaraswamy, Craig Schimmel, Gautam Biswas, Xenofon Koutsoukos, and Daniel Mack, “Vehicle Integrated Prognostic Reasoner (VIPR) 2010 Annual Final Report,” NASA/CR–2011-217147
6. M. Christensen, “Boeing 787 Central Maintenance Computing Function Summary,” Technical Report, Honeywell Labs., March 2010.
7. R.M. Button and A. Chicatelli, “Electrical Power System Health Management”, In *Proc. 1st International Forum on Integrated System Health Engineering and Management in Aerospace*, Napa, CA, November 2005.
8. G. Biswas, R. Kapadia, and X.W. Yu, “Combined Qualitative–Quantitative Steady-State Diagnosis of Continuous-Valued Systems,” IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans, vol. 27, no.. 2, pp. 167-185, March 1997.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 04 - 2013	**2. REPORT TYPE** Contractor Report	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Vehicle Integrated Prognostic Reasoner (VIPR) Metric Report	**5a. CONTRACT NUMBER** NNL09AD44T
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Cornhill, Dennis; Bharadwaj, Raj; Mylaraswamy, Dinkar	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 534723.02.03.07
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, Virginia 23681	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/CR-2013-217978

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 06
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES
Langley Technical Monitor: Paul S. Miner

14. ABSTRACT
This document outlines a set of metrics for evaluating the diagnostic and prognostic schemes developed for the Vehicle Integrated Prognostic Reasoner (VIPR), a system-level reasoner that encompasses the multiple levels of large, complex systems such as those for aircraft and spacecraft. VIPR health managers are organized hierarchically and operate together to derive diagnostic and prognostic inferences from symptoms and conditions reported by a set of diagnostic and prognostic monitors. For layered reasoners such as VIPR, the overall performance cannot be evaluated by metrics solely directed toward timely detection and accuracy of estimation of the faults in individual components. Among other factors, overall vehicle reasoner performance is governed by the effectiveness of the communication schemes between monitors and reasoners in the architecture, and the ability to propagate and fuse relevant information to make accurate, consistent, and timely predictions at different levels of the reasoner hierarchy. We outline an extended set of diagnostic and prognostics metrics that can be broadly categorized as evaluation measures for diagnostic coverage, prognostic coverage, accuracy of inferences, latency in making inferences, computational cost, and sensitivity to different fault and degradation conditions. We report metrics from Monte Carlo experiments using two variations of an aircraft reference model that supported both flat and hierarchical reasoning.

15. SUBJECT TERMS
Aircraft health; Diagnostics and prognostics; Evaluation measures; Performance metrics; System-level reasoner

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER ***(Include area code)***
U	U	U	UU	46	(443) 757-5802