

A system for fault management and fault consequences analysis for NASA's Deep Space Habitat

Silvano Colombano*, Lilly Spirkovska*, Vijayakumar Baskaran**, Gordon Aaseng*, Robert S. McCann*, John Ossenfort**, Irene Smith**, David L. Iverson*, Mark Schwabacher*

*NASA Ames Research Center

**Stinger Ghaffarian Technologies Inc.

Abstract

NASA's exploration program envisions the utilization of a Deep Space Habitat (DSH) for human exploration of the space environment in the vicinity of Mars and/or asteroids. Communication latencies with ground control of as long as 20+ minutes make it imperative that DSH operations be highly autonomous, as any telemetry-based detection of a systems problem on Earth could well occur too late to assist the crew with the problem. A DSH-based development program has been initiated to develop and test the automation technologies necessary to support highly autonomous DSH operations.

One such technology is a fault management tool to support performance monitoring of vehicle systems operations and to assist with real-time decision making in connection with operational anomalies and failures. Toward that end, we are developing Advanced Caution and Warning System (ACAWS), a tool that combines dynamic and interactive graphical representations of spacecraft systems, systems modeling, automated diagnostic analysis and root cause identification, system and mission impact assessment, and mitigation procedure identification to help spacecraft operators (both flight controllers and crew) understand and respond to anomalies more effectively. In this paper, we describe four major architecture elements of ACAWS: Anomaly Detection, Fault Isolation, System Effects Analysis, and Graphic User Interface (GUI), and how these elements work in concert with each other and with other tools to provide fault management support to both the controllers and crew. We then describe recent evaluations and tests of ACAWS on the DSH testbed. The results of these tests support the feasibility and strength of our approach to failure management automation and enhanced operational autonomy.

1. Introduction

The scope of this paper is a description of the Advanced Caution and Warning System (ACAWS) – a fault management tool designed to alert both flight controllers and crew of system anomalies and to guide them in the process of fault isolation and repair. The description of the tool is followed by a report on tests that were conducted on the ground-based Deep Space Habitat, an analog for a system where this kind of automation will be crucial.

Four major components of ACAWS are described in sections 3.1-3.4. These are modules for a) Anomaly Detection, whose function is to learn how the system typically behaves and to inform operators if current system behavior is unusual; b) Fault Detection and Isolation, which uses a system model to diagnose components that have failed; c) System Effects, which identifies components that will be impacted by failures; and d) Graphical User Interface which presents system views and diagnostic information in appropriate flexible formats to ground operators and astronauts.

In section 4, we describe related work in Fault Management. Two tools in use at NASA: CRANS and SEaCLIF are compared with ACAWS with respect to differences in scope and functionality.

Finally, in section 5, we describe three evaluations of the ACAWS system that have been performed using the Deep Space Habitat (DSH). The first evaluation was part of Desert Research and Technology Studies (D-RATS) in September 2011; the second evaluation was part of the Autonomous Mission Operations (AMO) test in June 2012, and the last evaluation was part of the Mission Operations Test (MOT) in September 2012. These tests built upon each other and served to direct the development of the system.

We conclude with an overall assessment of the current state of the tool and with plans for future work.

2. The ACAWS system: goals and philosophy

The Advanced Caution and Warning System (ACAWS) is a fault management tool that provides the following capabilities:

- Dynamic and interactive graphical representations of spacecraft systems
- Systems modeling
- Automated diagnostic analysis and root cause identification
- System and mission impact assessment
- Procedure and flight rule (FR) identification
- Interaction with other tools to help spacecraft operators (both flight controllers and crew) understand and respond to anomalies more effectively.

Each of these capabilities provides critical support for monitoring the performance of vehicle systems and real-time decision making by Mission Control Center (MCC) flight controllers and crew when faced with spacecraft anomalies and failures.

In addition to real-time mission support, ACAWS' capability to create and interact with malfunction scenarios offer significant opportunities to support the analysis and training requirements of crewed missions. Thus, the goals of the ACAWS development task are:

- Develop the technologies to support vehicle operators as they plan for, train for, and fly a spacecraft mission.
- Develop an infrastructure that allows reuse and integration of multiple products, enabling the operator to focus on accomplishing mission tasks with minimal need for managing multiple software tools.
- Understand what the operators' needs are, including but not limited to the following: a) what and how existing MCC tools can be integrated, b) what Integrated System Health Management (ISHM) technology can be used as is and what needs to be extended, and c) what is an effective concept of operations that incorporates ISHM technologies.

The product of the task is not just a prototype system. Equally important are the associated lessons learned in developing it.

The focus of ACAWS is on the needs of both flight controllers and onboard crew. Although we expect flight controllers to continue to assist a crew in low-Earth orbit in dealing with system malfunctions, for future deep-space missions the crew will need to accomplish some tasks autonomously due to communication time delays. We expect that providing similar tools to the flight controllers and the crew – albeit perhaps with a different level of detail and different display formats or interaction methods – could enable more effective and efficient collaboration between the two groups as well as heightened situational awareness. In the remainder of this paper, the word *operators* is used to refer to either flight controllers or crew.

3. Major ACAWS Components

ACAWS encompasses four major modules: anomaly detection, fault detection and isolation, system effects analysis, and a Graphical User Interface (GUI). The modules communicate with each other and with the DSH via a data distribution middleware; in this case, we use the Internet Communications Engine (ICE) [1] middleware. Additional modules are currently under development, depicted in the ACAWS architecture diagram shown in Figure 3.1. They will not be described in this paper.

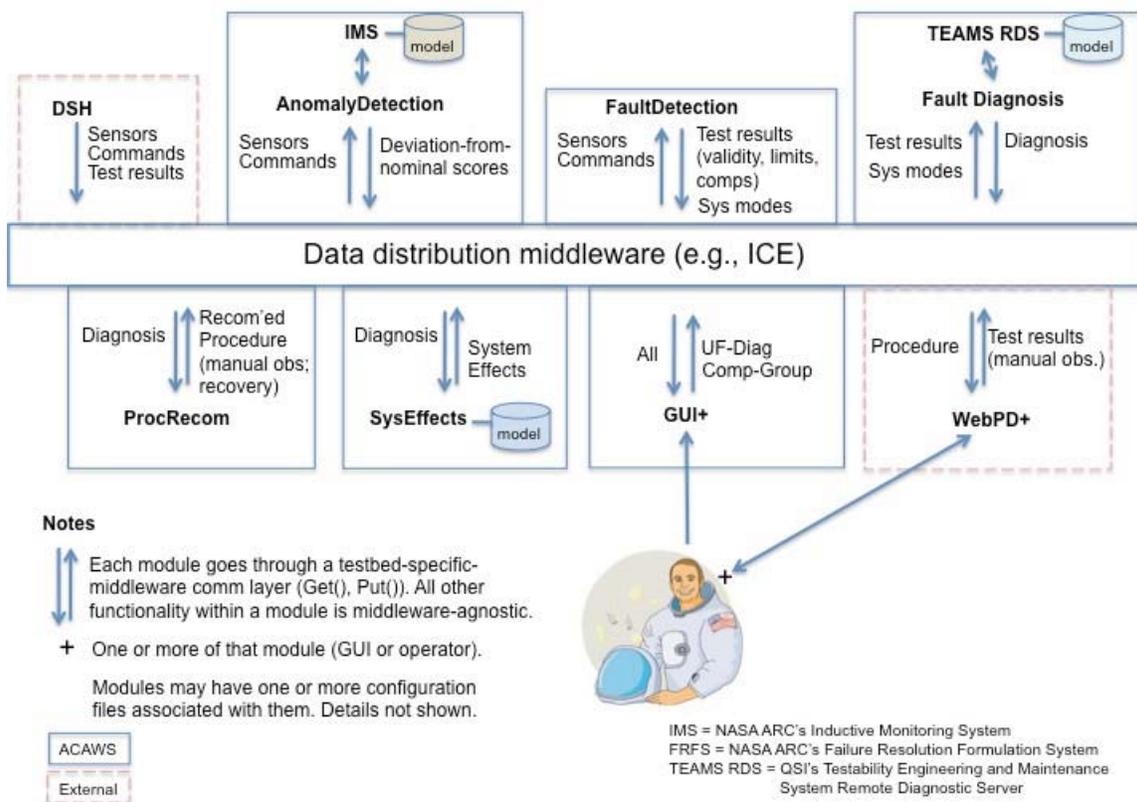


Figure 3.1: ACAWS architecture. Modules surrounded by dashed line rectangles are external to ACAWS.

3.1 The Anomaly Detection Module

The Anomaly Detection module uses the Inductive Monitoring System (IMS) [2a,b] to automatically learn how the system typically behaves and to inform operators if current system behavior is unusual. An IMS application has two pieces: training and monitoring. Training is done off-line and monitoring can be done onboard (on-line) or off-line. The goal of the training step is to learn how the system normally behaves. The input is a data set representing nominal system operation. The output of the IMS training process is a list of clusters that define nominal operations. A cluster is defined as a rectangular N-dimensional box (where N is the number of parameters used for training), with the cluster encoding the two extreme points of the box. IMS is trained off-line but can be retrained if system characteristics, and thus expected behavior, change, or as additional data is collected.

The goal of the monitoring step is to determine if the system is behaving differently than during training. The monitoring step extracts the relevant parameters from the incoming real-time system data (via the ACAWS Anomaly Detection module), normalizes and weights them accordingly, as was done during training, and then compares the incoming vector to the list of clusters generated during training. It outputs deviation scores, the distance between the vector and the nearest point contained in a cluster for the vector as a whole (a composite score) and for each parameter separately. These deviation scores, indicating how distant current conditions are from conditions categorized as nominal during training, are sent back to the communications middleware and are displayed on the ACAWS GUI.

IMS follows a data driven approach and does not require a hand-built model. Although helpful for some training tasks, a system expert is not required in developing an application. Moreover, IMS is trained on only nominal system data; it does not require failure data. For situations in which archived data or a high-fidelity simulator is available, IMS provides opportunities to develop a monitoring capability for complex systems that may be too difficult to characterize with model-based and rule-based systems.

3.2 The Fault Detection and Isolation Module

The ACAWS fault isolation module utilizes TEAMS-RDS [3] – one component of a suite of tools developed by Qualtech Systems Inc. (QSI) with support from NASA's Small Business Innovative Research (SBIR) program. The TEAMS suite of tools supports systems engineering, systems design and testability, automated diagnosis, and guided troubleshooting. TEAMS is a model-based reasoning system, meaning that relationships among various failure modes and readings from system instrumentation (observations) are captured in a model used by a run-time diagnostic engine. Below we describe a recent extension of our modeling work in which the TEAMS model forms the basis of failure consequence analysis to provide operators with insight into the system effects of a

component fault. This System Effects module can also be used in “what-if” mode, allowing the operator to determine in advance the effect of possible faults and malfunctions.

For real-time diagnosis, a dependency matrix (D-matrix) is generated from the model. The D-matrix is a two-dimensional matrix of failure modes (causes) and effects (“tests”; things that can be observed). Input to TEAMS is a vector of binary (true/false, pass/fail) health status indicators (“tests”), as computed by the DSH software and supplemented by the ACAWS fault detection module. DSH software provides observations on whether telemetry parameters are valid and whether they are in bounds. ACAWS-fault detection supplements these observations with heartbeat data providing observations of when data was last received from a component.

The TEAMS-RDS diagnostic engine uses system observations (e.g., whether a sensor value is within (“pass”) or exceeds (“fail”) a predetermined threshold) and determines which components could cause the current pattern of pass/fail system observations. The fault isolation module manages connection to the communication middleware (ICE) to retrieve the real-time telemetry or DSH-computed system observations, marshals the data into the appropriate form, sends it to TEAMS, and posts the diagnosis back to ICE.

ACAWS also exploits the TEAMS guided troubleshooting capability. Systems designers often cannot afford to (or choose not to) provide adequate instrumentation to fully disambiguate the cause of every failure. When the observations signature of a system failure cannot be fully disambiguated with telemetered (sensed) observations, TEAMS produces an ambiguity group, that is, the group of potential diagnoses that explain the failure signature. When an unambiguous failure is determined, ACAWS recommends a procedure to recover functionality (if possible) or to work around the loss.

The ACAWS recommended procedure is published on ICE and received by a procedure display tool – WebPD – developed at NASA Johnson Space Center (JSC), which the operator uses to step through the procedure instructions. The troubleshooting procedure ends with a request for the manual observation. This observation is sent, via ICE, to ACAWS where it is used for disambiguation. The loop continues until an unambiguous diagnosis is determined or ACAWS runs out of recommended tests for the operator to perform. The telemetry values, diagnosis, and recommended procedures are each displayed on the ACAWS GUI described in section 3.4.

3.3 The System Effects Module

The purpose of the System Effects (SysEffects) module is to determine what components will be impacted given a particular component fault or malfunction. “Impact” is defined as some change in functionality due to the absence of a needed resource (e.g. power), where the component itself is still functionally intact, i.e. capable of resuming its normal functionality if the needed resource is restored. A light bulb that is dark due to a power black out is “impacted”. This is different from a bulb that has burned out (i.e. “faulted”).

Note that the words “fault” and “failure” are used interchangeably for the purpose of this paper.

Our approach to building the SysEffects module was to make use of existing systems models as opposed to creating new ones. We felt that building separate models for diagnosis and failure consequences would be prone to inconsistencies, so we used the TEAMS diagnostic model as a starting point.

Unfortunately for our purposes, TEAMS models link different components for the purpose of building a dependency matrix that links faults to test points, as opposed to establishing a causal chain from component to component. The TEAMS model structure is thus necessary but not sufficient to provide the framework for SysEffects reasoning. To enable SysEffects, each component in the TEAMS model needed to be classified as “failed”, “impacted” or “nominal”. A classification of “failed” is provided by the diagnostic system. If a given component has been declared faulted, the system needs to determine whether an “impact signal” should be spread or not. If a “downstream” component receives an impact signal, the system needed to determine whether this component was actually impacted or not (there are components that receive an impact signal but, because of their particular functionality, are not actually impacted). These various determinations were embedded in Boolean logic inserted in the model components, within text fields assigned to that purpose, based on the modeler’s knowledge of the system and of the functionality of particular components. The behavior of different components is shown in Figure 3.2 and Figure 3.3.

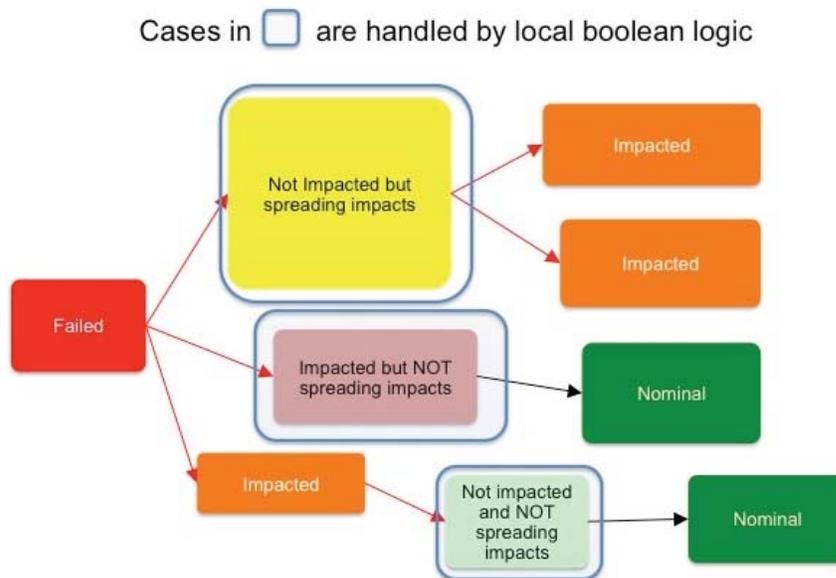


Figure 2.2: TEAMS model needs to be extended to enable SysEffects.

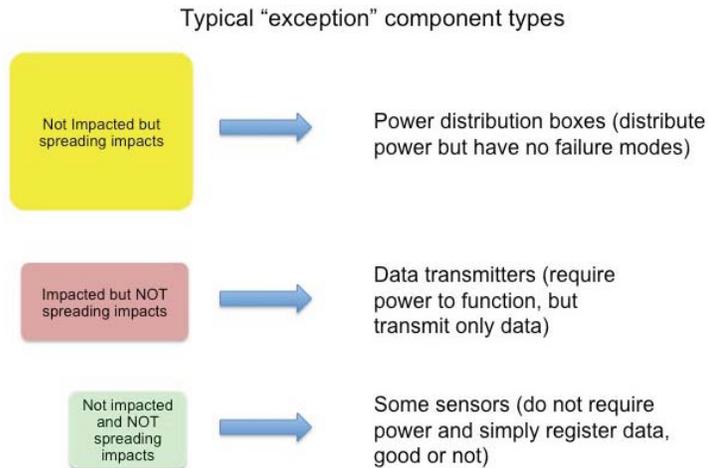


Figure 3.3: Key for component types that need to be handled as exceptions.

Ignoring syntax details, this is how these Boolean expressions are used in the algorithm described in the next section:

1. If the component is faulty, the Booleans indicate the following
 - a. The identified output or outputs should be set to FALSE
 - b. This component should not be put on the impact list
2. If the component is impacted, the Booleans indicate the following:
 - a. The identified output or outputs should be set to same values as the inputs – i.e. this component is “pass-through”, i.e. it will spread the impact – or it should be negated – i.e. this component will not spread the impact.
 - b. This component should be put on the impact list – or not
3. For all other components, all Boolean expressions are ignored.

The basic impact propagation algorithm described above is complicated by some additional features of TEAMS models: modes, fault uncertainty and the possibility of multiple faults. These features are handled as explained in sections 3.3.1-3.3.3.

3.3.1. Mode-aware impact assessment

It is typical of a system to be operating in different modes, each with different diagnostic strategies and outcomes. Essentially each mode requires a different system model. Instead of changing models for different modes, TEAMS uses specialized modules called “switches” that change the flow of signals as required by different system modes. These “switches” are not actual physical components, rather model devices that represent different types of operations. Switches are represented by boxes with one input and two different possible outputs, or two different possible inputs and one output. For example, in the DSH model, the ability to use either the Primary or Secondary 24VDC source is captured by way of a switch consisting of 2 input ports and one output port. The two input ports of the switch are connected to the corresponding PDU-B1 ports that represent 24VDC Primary and Secondary sources. When the system uses a primary source we are in particular mode, and when the system uses instead a secondary source we are in a different mode. The setting of these switches determines which mode we are in. Any

particular mode can be determined by the setting of several switches.

The SysEffects software extracts the operation modes and translates them into propagation Boolean expression for each of the affected switches. Since TEAMS switches do not actually represent a physical they are not included in the impact list.

3.3.2 Certainty dependent fault handling:

In order to handle ambiguity groups in TEAMS, the concept of "certainty" of faults was introduced. In the most recent implementation, a component fault can either be in the "Confirmed Failed (CF)" or "Possibly Failed (PF)" category. As the name suggests, a fault tagged as "CF" is one that TEAMS can tell for sure has failed based on the test results. In the event that TEAMS is unable to identify faults unambiguously, it has the ability to narrow down to a small subset of faults (ambiguity group) each of which will then be tagged as "PF". By definition an ambiguity group will consist of more than one possible fault.

3.3.3 Handling multiple faults:

In scenarios where TEAMS is able to pinpoint multiple faults as confirmed failures or when the best it could do is to generate an ambiguity list of candidate faults, it will be necessary for the system effects application to perform graph search for each fault and then aggregate the impact list generated by each search step. The aggregation step used to combine impact list for multiple faults depends on the "certainty" of the faults. In the case where all the faults are tagged as "CF", the aggregation step is simply the union of the impact lists of each confirmed failure mode. This union could be used to assist the operator with next-worst-failure analysis. If we have an ambiguity group where each fault is tagged as "PF", the aggregation step yields two lists of impacted components: the first is an intersection of the individual impact lists and the second is the cumulative list of all the impacted components without their intersection. The intersection would be used to assist the operator in determining what system components have been lost regardless of how the ambiguity gets resolved.

3.3.4 SysEffects software structure

The SysEffects software consists of three main components:

1. TEAMS diagnostic model parser
2. SysEffects reasoner
3. Communication mechanism that connects SysEffects to other modules in the system such as the ACAWS Fault Detection and Isolation module and the ACAWS GUI.

The front-end parser reads and converts an XML¹ version of the TEAMS model into a

¹ XML = Extensible Markup Language, a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

SysEffects representation that is primarily a graph whose nodes are TEAMS "nodes" that are made up of hierarchical components, failure modes and switches. The SysEffects reasoner is a depth-first graph search mechanism that starts its traversal from the graph node corresponding to the failure mode whose impact needs to be determined. The reasoner maintains an internal stack to hold the components being processed as well as a separate impact list that will be populated with the components that have been determined to be impacted. The reasoner publishes the final impact list when all affected nodes in the graph have been traversed. The communication mechanism enables the SysEffects application to service requests for impact analysis from either the ACAWS diagnosis engine, as will be the case during live operation, or from the ACAWS GUI during a user-fail mode system analysis. After performing a graph search as described in the previous section the results are broadcast into the communication layer and are picked up by the ACAWS GUI.

The SysEffects software components are generic and can be reused for other NASA programs. The model is specific to the system of interest and needs to be developed with SysEffects (and the operator displays, as described later) in mind or enhanced after the fact.

3.4 The Graphical User Interface (GUI)

Design of the ACAWS GUI is based on the *Model View Controller* (MVC) software architecture. The controllers manage the data aspect of the components, and are responsible for data retrieval, processing and mapping, and accepting the operator's inputs. The views are responsible for updating display panes and rendering the components. The views also respond to the controller's requests. The models hold data, control, view and auxiliary information together to notify the view to update when the state or data changes. The purpose of using such an architecture is to separate the view from the model so they can independently change or be modified, in a one-to-many relationship. For example, when the model logic changes, the view part of the code does not necessarily need to change. One model can have multiple views, which can help the operator understand different aspects of the model. There are multiple types of window panes in the framework to display and take operator action on different data types. Each of the window panes utilizes the MVC architecture. For example, the telemetry data provides sensor values that can be displayed on the sensor data pane. The same data can also be displayed on the block diagram but using a different view, in this case, just whether the data is available or missing, not the data's value. One model with two views can be realized easily with this MVC architecture.

The ACAWS GUI was developed in Java. Display screens were programmed in Java Swing, and an open-source software package – *MyDoggy* – that provides window allocation and docking capabilities. *Docking* means that a window can be dragged/dropped on a "docked" location and stay there relative to other windows, where it will move, resize, etc. along with those windows. When it is undocked, the window can once again be manipulated individually. Some customized work has been done to add

features such as saving the configuration of the working environment, docking multiple windows around a main window, and partitioning different areas of windows to group them together. The ACAWS GUI framework simplifies dynamic management of multiple working windows at run time. The ACAWS GUI has multiple types of window panes to display different types of information with different kinds of user interaction. All types of window panes are derived from the *dockable* framework so that all windows are dockable and manageable in the same fashion.

One of the key objectives for the general framework of the interface is providing flexibility to support the operator to work how she/he wants rather than dictating a certain approach.

The ACAWS GUI was used by flight controllers in the NASA JSC Mission Control Center (MCC) Operations Test Facility (OTF) as well as the crew in the DSH. Note that each of the ACAWS modules is independent of the GUI, connected only via the communications layer. Going forward, this independence makes it possible to easily integrate ACAWS modules with, for instance, Orion displays currently being designed.

4. Related work

There are two other major Failure Management systems in use in MCC: the Configurable Real-time Analysis System (CRANS) tool and System Effects and Capability Losses From Inserted Failures (SEaCLIF).

The major difference between CRANS [4] and ACAWS is that CRANS is a rule based Expert System, whereas ACAWS reasons from an actual system model. CRANS decision making is based on rules created and maintained by the operators, based on their experience, extensive studies of the vehicle architecture and failure mode documentation and simulation. “What if” scenarios can be followed as well. CRANS has been used for the International Space Station (ISS), and that experience indicated that keeping the system updated is a challenge, as changes are constantly being made and their consequences are not being derived automatically from documentation and reference manuals. In Mission Operations these are maintained by a “reconfiguration” group (RECON), and form the basis for all flight and ground operations and training. A quick and verifiable derivation of CRANS logic files from RECON products would be highly desirable. The aim of ACAWS is to address this shortcoming by using the actual spacecraft models for failure analysis.

CRANS is able to find single causes from multiple failures by analyzing a logic tree, built by the users, where items are related to each other by Boolean operators. The user can declare these items as on or off, failed or nominal, and see the effect of these settings on other items. The outputs are in the form of user defined matrices of color boxes representing the item or groups of items in the logic tree. The program is written in C, with development begun in 1992.

SEaCLIF, by J&P Systems Engineering [5], is a tool for integration of classic engineering data, such as certification limits and detailed design drawings, with operations data, such as crew procedures and flight rules, and with SR&QA (Safety, Reliability and Quality Assurance) engineering products, such as Hazard Reports, Failure Modes and Effects Analyses.

Integration is provided via a logic network that captures relationships between vehicle or system components, requirements, and hardware or software needed to implement the requirements. The logic network also captures relationships between hardware and software modules required for system control, hazards and associated causes, and power generation and distribution.

SEaCLIF provides the capability to analyze cross-subsystem effects due to component failures or state changes, and provides hot-links to existing web-based data bases. It is currently in use by NASA JSC's Safety & Mission Assurance (S&MA) Directorate in support of flight operations. The fundamental difference between SEaCLIF and both ACAWS and CRANS is that SEaCLIF is not a real-time time system, rather a tool for information management and data integration and analysis. In particular, it enables extensive “what if” reasoning, based on system simulation, looking at fault scenarios and consequences for mission objectives.

5. The Deep Space Habitat (DSH) ACAWS Evaluations

Three evaluations of the ACAWS system have been performed using the DSH: as part of Desert Research and Technology Studies (D-RATS) in September 2011; as part of the Autonomous Mission Operations (AMO) test in June 2012; and as part of the Mission Operations Test (MOT) in September 2012.

Developing a diagnostic model for the full DSH is out of scope for this task. To select a reasonable subset of the system for ACAWS evaluation purposes, we considered a number of characteristics for a reasonable investigative domain. First, the diagnosis of selected failure scenarios needs to be complex, both in the failure and the annunciation of the failure. Some desirable characteristics of such a failure include the following:

- Requires substantial system knowledge to perform manually
- Has a failure signature that looks similar to the signature of other possible failures, thereby providing a potential ambiguity of which failure happened
- Requires analysis to determine the failure, but is not so complex that it is difficult to explain in a demonstration presentation

Second, the selected subsystem needs to have enough sensors to allow for some automated diagnosis capability, either an unambiguous diagnosis for which a recovery procedure would restore functionality, or an ambiguous diagnosis with a small group of possible failures that are explained by the telemetered data and possible operator observations that could further disambiguate to the actual (seeded) failure.

The last consideration for investigative domain selection was criticality of the (seeded) failure. We wanted a failure that requires immediate resolution. This would ensure that the crew would exercise ACAWS in an on-board capacity rather than deferring the problem to an MCC/OTF controller.

With these objectives in mind, we selected a subset of the power subsystem. A number of failures in the power system have similar signatures, leading to complexity in the failure and annunciation of the failure. Further complexity arises because loss of power results in loss of data that is needed to determine system state. It requires both sensor data and operator input to diagnose failures. It includes both hardware failures and software failures, and the ability to differentiate between the two. Finally, the failures are each very significant but not catastrophic, and their impacts vary from fairly isolated to fairly broad.

D-RATS: During D-RATS [6], the DSH was in the Arizona desert, connected via a simulated time delay to flight controllers in Houston. The main purpose of the D-RATS ACAWS tests was to mature ACAWS on a real system in an operational setting rather than on a well-behaved “sandbox” system. Additionally, three operators (flight controllers) evaluated it for workload, ease of processing/ease of operation, information display, display format features, potential additions to ACAWS, and evaluation of operations with time delay effects. Workload assessments and user opinions were very favorable. Lessons learned from D-RATS informed subsequent development.

AMO: The Autonomous Mission Operations (AMO) [7] project conducted an empirical investigation of the impact of time delay on today’s mission operations, and of the effect of processes and mission support tools designed to mitigate time-delay related impacts. ACAWS was part of the evaluation of technologies that could potentially facilitate autonomous DSH operations.

The same (seeded) DSH Electrical Power System (EPS) failure scenarios were used for AMO as for D-RATS. Based on lessons learned from D-RATS, the ACAWS failure detection and isolation module was made more robust. Additionally, a second diagnosis engine (the Hybrid Diagnostic Engine, HyDE [8]) was integrated into ACAWS to handle failures in a simulated water transfer system. The integration was straightforward and involved connecting HyDE to the communications layer (ICE) and adding another display pane to the ACAWS GUI to represent the water transfer system.

Flight controller comments following the test indicated that both workload reduction and a reduction in the need for coordination resulted from ACAWS: *“ACAWS provided useful direction for the crew, so there was little need for us to do anything other than concur.”* Similarly, crew comments were both positive and indicated that the tool allowed the crew to proceed more autonomously than in Baseline: *“ACAWS told me which procedure to work which the ground later confirmed but I had already completed the procedure.”*

The last quote raises important issues of ACAWS-related impacts on crew situation awareness and operational autonomy, particularly as they might apply to mitigating detrimental effects of time delay: *“The time delay had little impact because ACAWS ran most of the procedure. Since the ground and crew can follow ACAWS, it was pretty seamless. MCC and DSH were able to come to common agreement with ACAWS. MCC and DSH statused each other via voice calls and texting.”*

MOT: During the Mission Operations Test, four people (one an active member of NASA’s current astronaut corps, the others astronaut surrogates) lived inside the DSH at JSC for ten days. These crewmembers communicated with flight controllers in the OTF over a simulated time delay. The goals of ACAWS participation in the Mission Operations Test (MOT) were two-fold: (1) test ACAWS integrated with Intelligent Controls technology, and (2) test the ACAWS SysEffects module, including user-fail “what-if” scenarios.

The MOT version of the DSH EPS was modified to support ACAWS and Intelligent Controls technology testing. In particular, redundant power sources were added for both the 24 VDC power supply and the 28 VDC converter, as shown by the diagram in Figure 3.

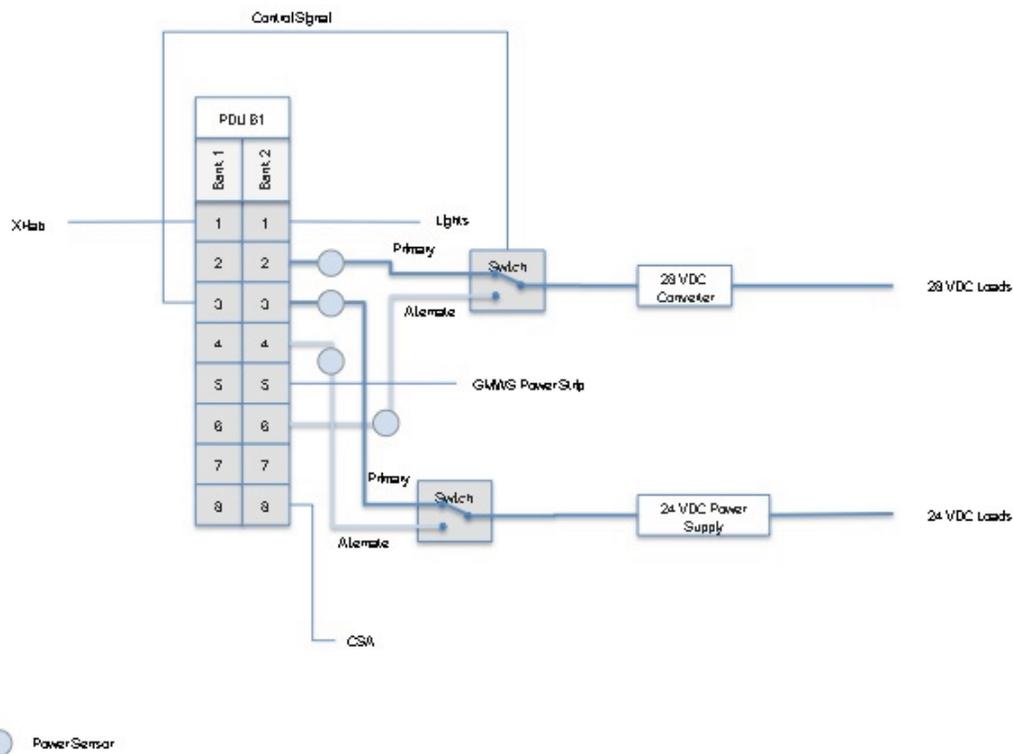


Figure 3.1: MOT DSH Redundant Power Configuration.

In the MOT tests, ACAWS was used to diagnose a simulation-injected failure in any of the four ports powering the converter or power supply. The Intelligent Controls software

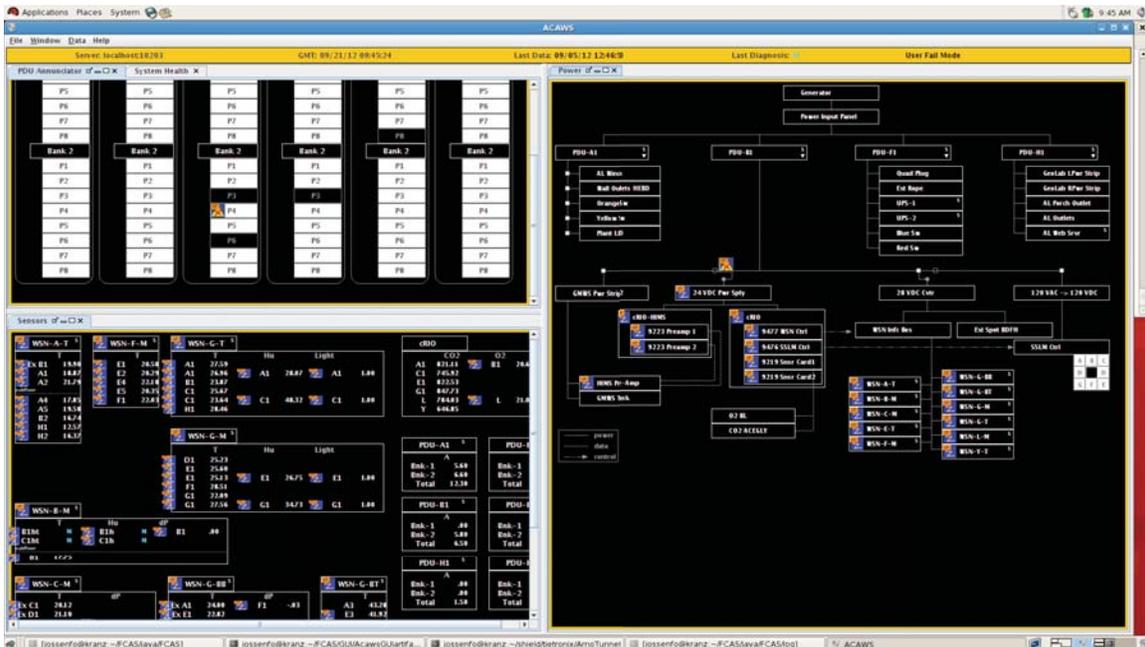


Figure 5.3: Offline "user-fail" GUI mode showing three views of operator-induced failure and its effects.

The figure illustrates the different symbology utilized by the GUI to distinguish real-time operations mode from off-line failure analysis mode. Note that the same failure is shown in both cases – a failure of one of the redundant ports powering the 24VDC power supply. In the “user-fail” GUI mode, the operator has “injected” that failure, as shown by the orange “UF” badge overlaid on the ACAWS confirmed-failed icon (stylized red “X” in a yellow triangle on a blue rectangle). The GUI transforms that induced failure into a SysEffects request which it posts to ICE. The same request message is used for real-time SysEffects requests, allowing the SysEffects module to be agnostic to failure source. The GUI matches the SysEffects reply with its request (since many operators could concurrently be performing “what-if” analyses) and displays those effects, similarly adding a “UF” badge to the ACAWS effects icon (stylized “E” in a blue rectangle). Moreover, the GUI status bar is orange and specifies “User Fail Mode”, as is the perimeter of each pane.

Off-line failure analysis mode, as designed for MOT, can be used for training. The trainee can inject any single failure of interest, hypothesize on the effects of that failure on the rest of the system, and then verify the hypothesis – and hence his/her system (e.g., DSH) understanding – via ACAWS.

6. Conclusions and Future Work

We addressed a need for greater automation in fault management with a tool designed to alert both flight controllers and crew of system anomalies and to guide them in the process of fault isolation and repair. This Advanced Caution and Warning System provides capabilities for a) dynamic and interactive graphical representations of

spacecraft systems b) systems modeling c) automated diagnostic analysis and root cause identification d) system impact assessment e) procedure and flight rule (FR) identification and f) interaction with other tools for more effective response to anomalies.

ACAWS has been tested and matured as a part of three technology evaluations in the context of the ground-based Deep Space Habitat. These were the Desert Research and Technology Studies (D-RATS) in September 2011, the Autonomous Mission Operations (AMO) test in June 2012, and the Mission Operations Test (MOT) in September 2012.

Future work is planned for all the modules described above. For the Anomaly Detection module this includes developing an *Alert Logic Filter* to filter out data spikes and issue alerts only when a specified number of sequential incoming vectors are anomalous. We will also be working toward automating the training step so that it can be accomplished without operator involvement. This would enable a “bare-bones” Anomaly Detection module to be deployed using minimal training data. As the system continues to operate nominally, the knowledge base could automatically be improved to incorporate that system behavior. This capability would increase the likelihood of IMS training data keeping up with acknowledged system behavior changes, either system characteristics – and thus expected behavior – change, or as additional data is collected.

There are many options for future work within the fault detection and isolation module. For example, QSI has been modifying TEAMS to provide additional capabilities based on our experiences. We will explore incorporating capabilities in both single-fault assumption diagnosis and in next-best-test recommendation. Many of our experiences have uncovered the need for a more robust fault detection layer. We will explore effective ways to characterize incoming system data and develop better data cleaning and filtering routines. We will also investigate effective ways to handle system redundancy, support multiple concurrent failures, deal with transitions from a failure state back to a nominal state, and incorporate contextual information such as phase-of-flight.

SysEffects capabilities, including handling of multiple failures and ambiguous diagnoses, need to be fully integrated with decision tools for mission operators. A different approach is also being explored with models that show cause-effect relationships more directly than done in the TEAMS tool. A software system called IDA (Interactive Digraph Analysis) developed by one of the authors is being tested for its ability to meet operator requirements which focus on component failures and consequences while still using TEAMS models, which are mainly based on relating failures to test points.

Currently, we are extending ACAWS to allow more complex user-fail queries, including setting the state of the system (e.g., turning ports ON/OFF, selecting which redundant path is active), composing multiple failure scenarios, and performing hybrid real-time/user-fail failure analysis. These extensions will support more complex training scenarios and enable operators to perform next-worst-failure analysis, that is, determining the additional failure (out of the myriad that are possible) that would maximally impact current crew activities or mission safety. In terms of the GUI, future work includes extending it to support complex failure analysis queries, restructuring it to more easily

support transition to a different system, and enhancing the symbology to reflect additional ACAWS capabilities.

7. References

[1] <http://www.zeroc.com/ice.html>

[2a] David L. Iverson, Rodney Martin, Mark Schwabacher, Lilly Spirkovska, William Taylor, Ryan Mackey, J. Patrick Castle, and Vijayakumar Baskaran. "General Purpose Data-Driven System Monitoring for Space Operations." *Journal of Aerospace Computing, Information, and Communication*. 9(2):26-44, October 2012. <<http://pdf.aiaa.org/getfile.cfm?urlX=%2D%3CWI%277D%2FQKS%2B%2ES0%23OV0%20%20%0A&urla=%25%2ARH%25%230H%20%0A&urlb=%21%2A%20%20%20%0A&urlc=%21%2A%20%20%20%0A&urld=%21%2A0%20%20%0A&urle=%28%2A%22H%23%21P%3ACTQ%20%20%0A&urlf=%28%2A%22H%23%21P%3ACTQ%24%20%0A>>

[2b] Spirkovska, L., Iverson, D.L., Hall, D.R., Taylor, W.M., Patterson-Hine, A., Brown, B.L., Ferrell, B.A., and Waterman, R.D., "Anomaly Detection for Next-Generation Space Launch Ground Operations," *Proceedings of the AIAA SpaceOps 2010 Conference*, AIAA, Huntsville, AL, April 2010.

[3] <http://www.teamqsi.com/>

[4] McCluney, K. "CRANS - Configurable Real-Time Analysis System" *NASA Technical Report*, Jan 1994 <<http://ntrs.nasa.gov/search.jsp?R=19940003101>>

[5] <http://www.jandptech.com/innovations.html>

[6] http://www.nasa.gov/exploration/analog/desertrats/drats2011_wrapup-report.html

[7] J. Frank, L. Spirkovska, R. McCann, L. Wang, K. Pohlkamp, L. Morin. Autonomous Mission Operations. *Proceedings of the IEEE Aerospace Conference*, 2013.

[8] <http://ti.arc.nasa.gov/tech/dash/diagnostics-and-prognostics/hyde-diagnostics/>

Acknowledgements

ACAWS was funded by NASA's Game Changing Development Autonomous Systems project. We thank Ann Patterson-Hine for continued support of ACAWS development, Adam Campbell, Charles Lee, and Brandon Oubre for vital programming assistance, and the NASA JSC DSH team, NASA JSC flight controllers and trainers, and QSI's TEAMS team for invaluable collaboration.