

NASA-CR-191914

Document Number 00157A-AR-89

aura

1N-19-CR

141248

p. 22

RECEIVED

SEP 11 1991

PATENTS & TU OFFICE

Final Report on the Galileo PPS Expert Monitoring and Diagnostic Prototype

January 30, 1989

N93-21823

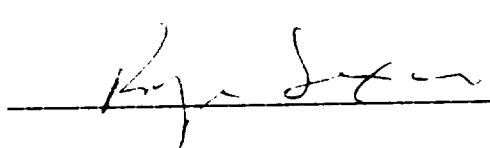
Unclass

G3/19 0141248

Received by: _____

Dr. Ted Bahrami
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Approved by: _____


Dr. Rogers Saxon
Aura Systems Inc.
6033 West Century Boulevard, Suite 720
Los Angeles, California 90045-6416

(NASA-CR-191914) THE GALILEO PPS
EXPERT MONITORING AND DIAGNOSTIC
PROTOTYPE Final Report (Aura
Systems) 22 p

Contract # 958346

Release Log

Rev	Mod	Description	Date	Auth
00	00	Original	1-30-89	SPB

PRECEDING PAGE BLANK NOT FILMED

This report was prepared for the Jet Propulsion Laboratory,
California Institute of Technology, sponsored by the
National Aeronautics and Space Administration.

Contents

Section 1	
Introduction.....	4
Overview	4
Purpose of Document.....	4
Section 2	
Monitoring and Diagnosis.....	5
Monitoring and Diagnosis Systems.....	5
General Models for Fault Diagnosis	
Expert Diagnostic Systems.....	7
Reasoning Strategies.....	7
Section 3	
Implementation Issues.....	9
Knowledge Acquisition.....	9
Knowledge Representation	10
Knowledge Tools.....	11
Development Strategy	13
Section 4	
Implementing the EMM Prototype	14
Selecting a Knowledge Tool.....	14
Reasoning Strategy.....	14
Designing the Knowledge Representation	17
Knowledge Acquisition.....	17
Section 5	
Conclusions and Recommendations.....	18
Glossary.....	20

Acknowledgements

The work described in this report was supported by JPL contract #958346. The authors are indebted to power system experts Terry Koerner and Dick Johnson for their invaluable assistance in creating the knowledge base. We also appreciate the guidance and design suggestions furnished by the JPL principal investigator, Dr. Khosrow ("Ted") Bahrami.

Section 1: Introduction

Overview

The Galileo PPS Expert Monitoring Module (EMM) is a prototype system implemented on the SUN workstation that will demonstrate a knowledge-based approach to monitoring and diagnosis for the Galileo spacecraft Power/Pyro subsystems. The prototype will simulate an analysis module functioning within the SFOC Engineering Analysis Subsystem Environment (EASE).

Existing spacecraft monitoring & diagnosis procedures require a human expert to manually sift through spacecraft telemetry and command sequences in order to verify spacecraft health and to identify the causes of anomalies and faults. The Expert Monitoring Module will permit more effective use of the expert's time by:

- automatically monitoring spacecraft health,
- verifying planned events,
- automatically diagnosing simple or frequently-encountered faults and anomalies,
- explaining the reasoning behind its conclusions.

Automated diagnosis can be implemented using traditional methods, or using more recent knowledge-based approaches.

Purpose of Document

This document describes the implementation of a prototype EMM for the Galileo spacecraft Power Pyro Subsystem. Section 2 of this document provides an overview of the issues in monitoring and diagnosis and comparison between traditional and knowledge-based solutions to this problem. Section 3 describes various tradeoffs which must be considered when designing a knowledge-based approach to monitoring and diagnosis, and Section 4 discusses how these issues were resolved in constructing the prototype. Section 5 presents conclusions and recommendations for constructing a full-scale demonstration of the EMM. A Glossary provides definitions of terms used in this text.

Section 2: Monitoring and Diagnosis

Monitoring and Diagnosis Systems

The purpose of a monitoring and diagnosis system is to verify events and to detect, classify and explain faults in a monitored device. The monitoring function receives data from the device, and is able to determine whether the device is functioning normally. If abnormal behavior is detected, the diagnosis function is invoked to classify, and possibly explain, the abnormal behavior.

A specification is transformed into a working device in three successive stages: design of the device, implementation of the physical device, and operation. When a divergence is observed between the behavior the operating device and our expectations, the cause can lie within one of the following areas:

- The design is not consistent with the specification.
- The implementation not consistent with the design.
- A component has failed.
- The device functions correctly, but our expectations are wrong.
- One or more of the above.

The first two causes assume a failure in the design or implementation stages of device development. In monitoring applications it is usual to assume that the design and implementation have been verified, and so the fault model is confined to the "operation" phase. Any diagnosis system implicitly incorporates further assumptions about fault analysis and about the device being diagnosed in order to make the diagnostic process manageable. For example, assumptions are made about:

- Device behavior as explainable in terms of the functioning its primitive components.
- The granularity of the device, or the level at which components are considered to be primitive (atomic).
- Single fault assumption (that the symptoms are caused by a single failure).
- Multiple fault assumption (the symptoms may have been caused by more than one failure).
- Nonintermittency of the failure source (the device behaves consistently over time) versus intermittent or transient faults.
- Functional, structural hierarchy of the device (whether the device forms a subcomponent hierarchy). Functional overloading may compromise the strict hierarchical structure.
- Whether there are feedback loops, which complicate the causal relationships between causes and symptoms.

- Steady-state failure modes (that failures eventually lead to a stable system state). If there are divergent failure modes, then the current system state will give little indication of the initial cause of the failure (for example, when a computer, through a program bug, begins executing data as instructions).

The choice of assumptions made in the fault model influences the coverage and resolution achieved by the diagnostic task.

The functional requirements for a monitoring and diagnosis system are also driven by the anticipated interactions among the user, the device and the diagnostic system. A monitoring and diagnosis system will at the minimum receive a real-time data stream from the device. The system may also act upon the device by requesting tests, or commanding the device to perform corrective or preventive actions. The system may interact with the user by querying for information, requesting tests, providing explanations or obeying user requests to modify its own functions. The system will also need to account for user-device interactions if the user makes inputs to the device which alter its state.

An expert system which performs a monitoring function must ensure that data is processed at the same rate that it arrives, otherwise the system will fall inexorably into arrears. The system will also incorporate procedures for screening the input data for errors. A diagnostic system which makes corrective control inputs to the device must ensure that these inputs are issued in a timely manner.

General Models for Fault Diagnosis:

There are two principal approaches to fault diagnosis, fault-based and model-based. A fault-based system has an explicit fault model, often in the form of a fault dictionary, specifying a mapping from specific faults to the induced behavior. Model-based systems predict behavior from the system design, explaining faults as deviations from expected behavior or vice-versa. These systems have a fault model which is implicit in the "depth" of the system model. It is possible, and often advantageous, to construct a hybrid diagnostic system which can apply either method according to circumstance.

Traditionally, fault-based approaches have been used for diagnosis. These methods are well understood, but difficult to realize for complex systems. More recently, sophisticated model-based approaches have been applied to diagnosis. The model-based approaches derive from

systems used for design verification, which predict the behavior of a design in order to verify that the design satisfies the specification.

The nature of the device under diagnosis influences the choice of assumptions and feasibility of a model-based approach. For example:

- A functional or structural simulation of a biological system (e.g. human) may be impractical, and seem absurd to the practitioner.
- Processing time may be a critical factor, as in real-time systems. For example, efficient algorithms are available to simulate digital circuits, but algorithms to simulate analog effects (gate fanout, bus loading, etc) are extremely expensive.
- In remote monitoring applications, sensor data may be imprecise, unreliable, or insufficiently complete.
- In time-critical applications, data may be valid only for a limited duration, requiring capability for temporal reasoning.
- In time-critical applications, the ability to focus attention according to priorities will be a requirement.

Traditional diagnostic systems are generally fault-based. Knowledge-based diagnostic systems may incorporate both fault-based and model-based diagnostic methods, and are often a hybrid of both approaches. The following section will discuss the strengths and weaknesses of the various types of expert diagnostic systems in light of the foregoing considerations.

Expert Diagnostic Systems.

An expert diagnostic system which interacts with the user by providing explanations must have the facilities to understand questions and provide useful answers appropriate to the user's expertise. If the system will receive data from the user then the system needs to verify the accuracy and consistency of this data.

Reasoning Strategies

Diagnostic reasoning means finding a process model to solve the problem: given a set of symptoms, identify and explain the cause of the abnormal behavior. Reasoning strategies can be loosely assigned to one of three categories: deep reasoning, shallow reasoning, and hybrid. The term shallow reasoning implies a method using a relatively superficial understanding of the problem domain, such as rules-of-thumb and rote procedures. In contrast, deep reasoning implies the use of a device model which uses basic principles and detailed knowledge of the

structure and behavior of device components to predict normal behavior of the device. A hybrid system will optimize for efficiency by selecting features from both approaches, often using a hierarchical strategy.

Shallow Reasoning

For diagnostic applications, approaches which fall under this heading are fault dictionary and/or decision tree methods, and shallow causal reasoning. These approaches require only a knowledge of system behavior under failure. They do not directly support the generation of good explanations.

Fault Dictionary Models: A fault dictionary is a shallow model of the device misbehavior. This method requires explicit enumeration of all possible faults. The fault diagnosis amounts to table-lookup or tree-search in a faults-vs-symptoms database. Though this technique is termed "shallow", the fault dictionary may have been developed from a sophisticated model of the device, with the possible failures compiled into a fault-tree to optimize searches at run-time. Typically, failures are injected into a simulation, generating signatures at designated test points that are recorded in a table. A single-fault assumption (SFA) is usually made.

Decision Tree: A decision tree is a more add-hoc approach to encoding diagnostic knowledge. It is well suited to problems having a hierarchical decomposition, where a node in the tree represents a decision procedure to isolate the fault to one of several subnodes. The decision tree need not enumerate all possible faults, and it is more amenable to the expression of heuristic knowledge to optimize the efficiency of fault isolation and classification.

Shallow Causal Reasoning: This approach might be described as using "rules-of-thumb" to hasten problem solving, and is often used to screen out simple problems before invoking a more expensive deep analysis, or to isolate a fault to some subcomponent of the device. As with the fault tree method, search can be optimized to look for common or critical faults first, but faults need not be enumerated in advance. The diagnostic process starts with the observations and forms a set of hypotheses. The hypotheses are then subject to rejection, verification and refinement. Users may be cued to run tests and provide results or other evidence.

Shallow causal models attempt to capture the field experience of an expert through associations between observations and hypotheses. The expert knowledge is used to select a limited number of likely hypotheses to pursue, rather than working out all possible cause-effect relationships that might relate a symptom to its antecedents and consequences. No reason for the linkage is required

(hence the shallowness), and therefore such systems are limited in their ability to justify their results. If the system is to detect misbehavior automatically, then the problem of specifying normal behavior must also be addressed. This approach was felt to be the most appropriate for the Galileo Power/Pyro diagnostic system. We had access to a domain expert with many years of field experience who was able to provide useful heuristics. Additionally, we had to consider the need to perform diagnosis in real-time, a constraint which argues in favor of the relative efficiency of the shallow reasoning approaches. As noted above, the shallow-causal reasoning approach does not address the problem of specifying normal behavior of the system. Thus, we had to plan on creating a separate model of the Galileo PPS to predict nominal telemetry values.

Deep Reasoning

Shallow causal reasoning describes the first generation diagnostic systems. The second generation models the structure and/or the behavior of the device. Structure modeling refers to a description of the design topology, or component connections. Functional modeling, by contrast, means to simulate the behavior of the device. In many cases this is impractical to do using a purely rule-based diagnostic system. Misbehavior can be detected automatically by comparing observed behavior to the expectations generated by the device model. Diagnosis then consists of tracing discrepancies down to a primitive subcomponent. This technique derives from systems which perform design verification by comparing the predicted behavior of a design to the device specification.

Hybrid Reasoning: Hybrid reasoning strategies are a combination of the above techniques. Hybrid reasoning is generally adopted either for efficiency (shallow reasoning first, deep later) or for generality or both. Such systems vary in the degree of interaction between the shallow and deep reasoners. In the case of Galileo Power/Pyro, the shallow reasoning implemented in the prototype may be augmented with model-based reasoning for certain classes of faults, resulting in a hybrid system.

Section 3: Implementation Issues

Knowledge Acquisition

The acquisition of knowledge may be pursued via many sources, including human domain experts, schematic diagrams and operating manuals, existing simulation software and diverse other sources. The skill of interviewing human experts is very valuable to the knowledge engineer, since domain experts are seldom able to make a detailed exposition of all the knowledge that they

use to perform their work. The knowledge engineer must be able to organize the expert's knowledge, understand the problem solving methods which are used, and also look for omissions and inconsistencies which will require clarification. Knowledge acquisition systems have been demonstrated which are able to assist the knowledge engineer in encoding the acquired knowledge so that consistency and completeness checking can be performed automatically. One shortcoming of these systems is that they are generally not able to perform this function on the implemented knowledge base, and so do not support debugging and maintenance of the implemented system.

Knowledge Representation

There are five methods for representing knowledge which are in general use. These are:

- semantic networks
- object-oriented
- production rules, assertions
- logic programming
- frame based

Semantic networks are the most general representation scheme. Nodes may correspond to objects which may be physical or conceptual in nature. Links relate objects. For example, membership in a class may be represented by a link of type "is-a" between an instance object and a class object.

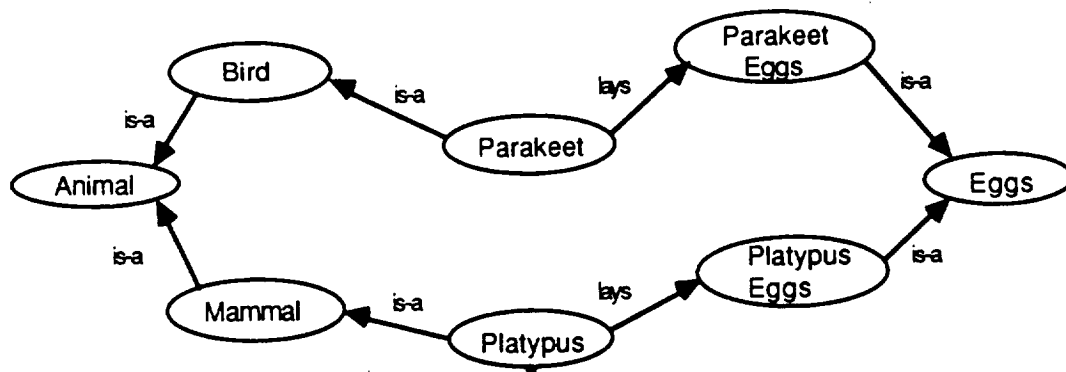


Figure 1. An example of a semantic network.

Object-oriented representations are an instance of a semantic net in which every object must be a member of a class, and instances of a class automatically inherit the attributes of their superclasses. This is a powerful conceptual tool; for example, if parakeet is defined as a subclass of the class of

birds, then any rules which apply to birds will apply to parakeets as well (but the reverse will not be true) and parakeets will inherit the attributes of birds, such as "Lays Eggs".

Production rules are used with facts or assertions, which are often object-attribute-value triplets. Object-attribute-value triplets may be seen as a special case of semantic nets in which links are either of type object->attribute or attribute->value. Nodes are either objects, attributes or values. Production systems typically do not support inheritance hierarchies.

A frame is a description of an object which contains slots for all of the object's attributes. Like attributes, slots may store values. In addition, slots may store default values, sets of rules, procedures for obtaining the slot's value, or pointers to other frames. Frames allow richer representations than the object-attribute-value scheme, but their added complexity is warranted only in situations where more powerful representation is required.

Logic programming is based on the model of predicate calculus. Knowledge is represented by expressions which are true or false. Thus, if one has a bicycle whose color is blue, then the predicate color(bicycle, blue) would be true. It would not possible to directly obtain the color of the bicycle, however, since blue is an independent object which participates in a "color" relation with the bicycle. We may only ask whether a color(bicycle,blue) relation exists in working memory. Logic programming is supported most notably by the Prolog programming language, and despite the apparent restrictiveness noted above, it is a very powerful language.

The knowledge representation will be chosen based on the reasoning strategy. Shallow causal reasoning will call for use of production rules, but a modeling approach using deep knowledge calls for use of a richer set of data structures and programming primitives, provided by object and frame-based environments. Having chosen to apply shallow causal reasoning to the Galileo PPS diagnostic problem, we sought to represent diagnostic knowledge in the production-assertion paradigm which is best suited to this form of reasoning.

Knowledge Tools

Knowledge tools are distinguished by their capabilities in the following areas:

- Reasoning strategy
- Knowledge representation
- Control algorithm, conflict resolution strategy

We have already discussed the various types of reasoning strategies and knowledge representations which may be employed. Knowledge tools for rule-based systems are characterized by a control algorithm, which may be forward-chaining, backward-chaining, or a combination of both, to control the execution of production rules.

Forward Chaining

A forward-chaining knowledge tool will provide a built-in control algorithm consisting of three steps which are repeated in a loop, known as the "recognize-act" cycle. The first step is to match facts in working memory against the patterns in the antecedent clauses of rules, keeping a list of all the complete instantiations found, where a complete instantiation consists of a rule and facts to match each of the patterns in its antecedent. This set of complete instantiations, referred to as the "conflict set", represents all of the valid inferences which may be drawn from working memory. The second step, conflict resolution, involves choosing which of these inferences to make first, since one inference may have the effect of making another one invalid. The third step is to execute the consequent clauses of the selected instantiation, which will have the effect of modifying working memory.

The conflict resolution strategy is a set of heuristic criteria which guide the selection of an inference from the conflict set. Common heuristics include "most recent match first" and "most specific match first"; some systems allow rules, or an instantiation of the rule, to specify their own priority (often called salience). The inference engine may have its conflict resolution strategy hardwired in, or it may provide hooks for the user to specify a resolution strategy.

Backward Chaining

A backward-chaining control algorithm looks at the consequent clauses of rules to determine if the rule can satisfy a posted goal. All rules which meet this criterion are examined in their antecedent clauses to determine if the rule represents a valid inference. Antecedent clauses which are indeterminate become new posted goals, which the control algorithm attempts recursively to satisfy. This algorithm provides a very natural means for expressing diagnostic knowledge, and so is commonly used for diagnostic expert systems (e.g., MYCIN). For example, a knowledge base for automobile diagnosis might include the rules:

(if battery-is-not-dead => not dead-battery-at-fault)
(if horn-does-work => battery-is-not-dead)

In the course of diagnosing the reason a car will not start, a backward-chaining control algorithm would try to determine if a dead battery is at fault. Based on the first rule it might attempt to determine whether the battery is dead. This new goal would bring the second rule into play, creating the new goal of determining whether the horn works. This regression must terminate somewhere; in this case it might terminate by asking the human operator to try to sound the car's horn, entering the result of the experiment. This type of goal-based reasoning behavior may also be realized in a forward chaining system through use of the hypothesize-and-test paradigm. In this paradigm, an antecedent clause of a rule specifies what goals the rule may help to satisfy. Since the forward-chaining control algorithm does not attempt to satisfy the other antecedent clauses of a rule, other means must be employed to steer the inference process in a useful direction.

Development Strategy

The development plan should include plans for

- knowledge acquisition
- implementation
- verification.
- maintainence.

The knowledge to be acquired will depend on whether the reasoning strategy is to be fault-based or model-based. If the modeling approach is used, or if the fault dictionary is to be compiled off-line, the deep knowledge about the device and behavioral models of its primitive components will be required. For a device which already has an operational history, the fault dictionary may be available in the form of operation manuals, logs, etc.

It is widely agreed that the optimum implementation strategy for an expert system consists of three phases:

- functional prototype: proof of concept, knowledge rep and problem-solving strategies.
- full-scale demonstration: implement full capability of anticipated system, verify correctness, adequate speed, explore user interface concepts.
- integrated online system: fully integrated system with user interface, manuals, etc.

Performance of fault-based systems can be verified by exhaustively exercising each entry in the fault-dictionary in a test-bed environment. Systems based on deep modeling require that a set of test-cases be designed which span the space of the fault model. The knowledge representation chosen in the implementation must be structured and documented in a way that enhances maintenance.

Section 4: Implementing the EMM Prototype

Selecting a Knowledge Tool

As noted previously, there are a variety of considerations which enter into the design of a knowledge-based system for monitoring and diagnosis. We felt that the driving consideration was the need to integrate with the EASE workstation environment. This dictated the use of a knowledge tool which is portable, can be extended by the user, and is capable of being integrated into other applications. The need to integrate with EASE also made CPU-efficiency an important goal. The EASE workstation will host a number of real-time telemetry analysis processes, and also provide a color-graphics interface to the user. Therefore, no analysis process may depend on getting more than a fraction of the CPU and other system resources. Also, we had decided to use a production-rule system, since this approach has been used successfully in diagnostic applications. The most efficient production systems employ the Rete algorithm to perform pattern matching and conflict resolution. The complexity of pattern matching increases exponentially with the size of the knowledge base, so it is crucial that this task be done efficiently. Production systems which are based on the Rete algorithm include OPS5, its successor OPS83, and the NASA-standard CLIPS. Of these, the CLIPS system also met our goals for portability and integration, since it is written in C and the source code is provided. Based on these factors, we chose CLIPS to implement the EMM.

Reasoning Strategy

The CLIPS shell can represent knowledge only through production rules and assertions. This can be constraining since, although it is possible to construct an object-attribute-value models using this representation, the object-oriented concepts of class-hierarchy and inheritance are not supported. This limitation would be more significant if we had chosen to employ deep reasoning, however, as was discussed previously, we had decided on shallow causal reasoning due to the requirement for real-time performance. Because shallow reasoning does not generally provide a model of nominal system behavior, we needed to provide a separate model to make telemetry predictions based on the command sequence which was sent to the spacecraft. The resulting dataflow in the EMM is depicted in Figure 2 below.

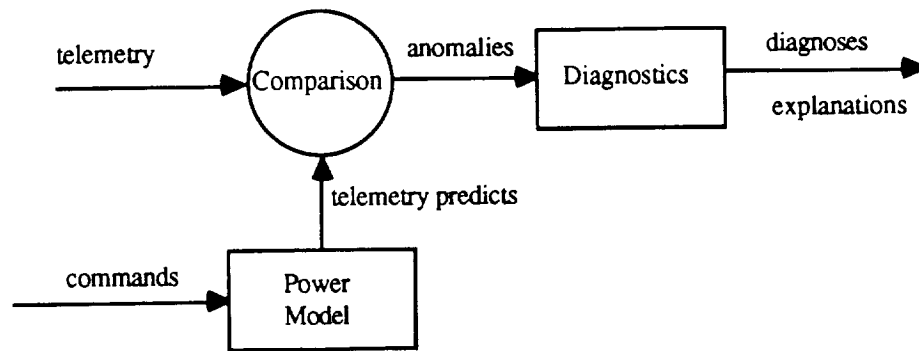


Figure 2. A dataflow diagram for the Galileo PPS Expert Monitoring Module

A second question was whether we would need incorporate uncertainty into our reasoning. We decided that this would likely not be the case. Systems which use uncertainty, such as MYCIN, typically must deal with evidence which is incomplete and qualitative in nature. This gives rise to problems where there are many competing hypotheses, which must be discriminated using heuristics based on probability and statistical factors. The Galileo engineering telemetry, by contrast, was designed to provide data that is quantitative, discriminating and partially redundant. Therefore we expect that cases where the telemetry is inadequate or untrustworthy will be infrequent. Based on this premise, we chose a reasoning paradigm which is simple and quite general: hypothesize-and-test. As we noted earlier, this paradigm is quite useful in producing goal-directed behavior in systems with forward-chaining control, such as CLIPS. In the hypothesize-and-test paradigm, various classes of faults are arranged as hypotheses in a hierarchical classification, as illustrated in Figure 3.

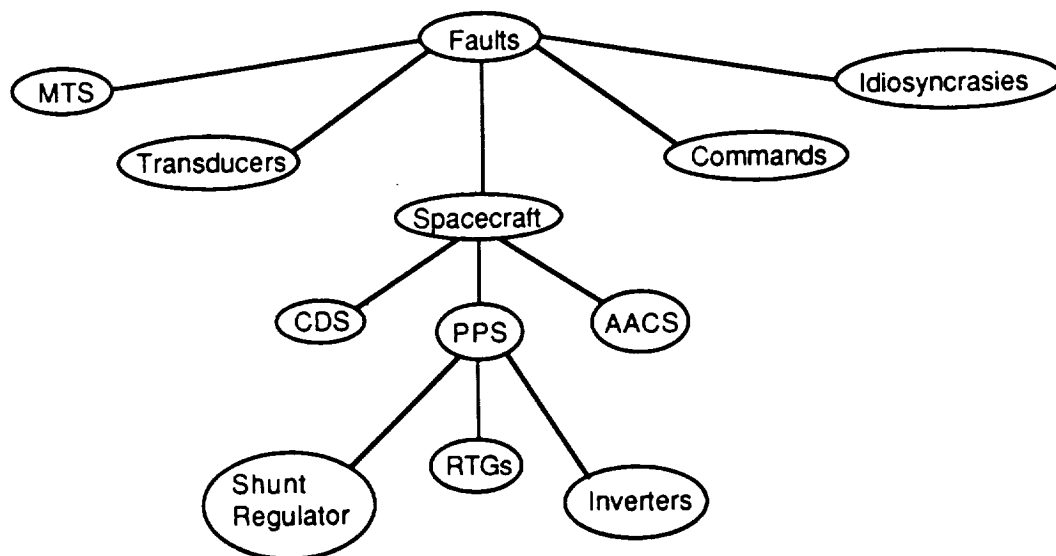


Figure 3. Hypotheses arranged in a hierarchy.

A given anomaly will suggest a set of hypotheses which are ordered by decreasing likelihood. Each hypothesis in turn will trigger rules to either accept, reject or refine it, where refinement consists of expanding a hypothesis into more specific hypotheses which are compatible with the observations. A hypothesis is accepted if it is consistent with the evidence and if it cannot be further refined. This method of classifying anomalies results in goal-directed search with heuristic pruning behavior in a forward-chaining environment.

Algorithm "Accept, Reject, or Refine":

- 1 If current hypothesis is indicated, and no refinement is possible,
Accept hypothesis.
- 2 If current hypothesis is contraindicated, Reject hypothesis.
- 3 While plausible refinements exist,
Apply "Accept, Reject or Refine" to refinements.

The explanation generator will produce a trace of the diagnostic search as it tests hypotheses one by one, similar to the example output below:

```
ALARM:  channel 69, the "+X RTG OUTPUT VOLTAGE", level 30.85, high by 0.10
ALARM:  channel 70, the "-X RTG OUTPUT VOLTAGE", level 30.88, high by 0.10
REFINE:  I see that channel 69 is in alarm. I assume that the anomaly is caused by one of the following possibilities:
          1) A transducer problem has caused a faulty reading;
          2) A telemetry problem has garbled the channel data;
          3) Some other anomalous condition caused a valid alarm condition.
REJECT:  The alarm in channel 69 is not due to a transducer problem because there are other channels in alarm also.
DIALOG:  Now considering the possibility that there is: telemetry-problem
REJECT:  The alarm in channel 69 is not a telemetry problem because there are other channels which are not in alarm.
DIALOG:  Now considering the possibility that there is: anomalous-condition
REFINE:  I see that channel 69 is in alarm. This could indicate an undervoltage-trip or a dc-bus-overload.
ACCEPT:  The alarms are due to an undervoltage-trip, because both RTG output voltages are too high.
```

The example above shows the rejection of the first two hypotheses, "transducer-problem" and "telemetry-problem", because there is contrary evidence. The third hypothesis, "anomalous-condition", is refined into the subsumed hypotheses "undervoltage-trip" and "dc-bus-overload". The first of these matches the observed symptoms, and is therefore accepted to be the cause of the anomaly.

Designing the Knowledge Representation

A relational model is best adapted to the data structure provided by CLIPS, using assertions of the form

```
(load "NIMS OPTICS HTR 1" type AC bus SECONDARY section SPUN value 39.6 state OFF)
```

In this example, the first field identifies the object as one of type "load" and the second field is a key which uniquely identifies this object. The rest of this structure consists of attribute-value pairs which are common to all objects of type "load". The antecedent clauses of a CLIPS rule perform pattern matching on these objects, for example:

```
(defrule is-there-an-AC-load-on
  (load ?name type AC bus ? section ? value ? state ON)
  =>
  (fprintout t "Load " ?name " is on."))
```

The pattern in this rule will match any load whose type is AC and whose state is ON, and each such match will create an independent instantiation of the inference. The symbol "?name" is a variable which is bound to the name of the load which generated the instance. The CLIPS inference engine uses refraction in conflict resolution to prevent the trivial loops which would arise from rules like this one, since the antecedent clause would otherwise match on the same load over and over again.

Knowledge Acquisition

The knowledge acquisition effort for the EMM was targeted toward a demonstration of depth but not to complete coverage of the fault space. Thus, the knowledge acquired is sufficient to support anomaly detection and to diagnose certain faults of representative complexity. The knowledge base may be partitioned into two domains, the first domain being modeling knowledge required to predict nominal behavior, and the second domain being knowledge required to classify faults.

The power modeling knowledge was acquired via interviews with JPL domain expert Terry Koerner, the author of the GPWR program, which is used to simulate the Galileo power system. The source code for this program was also a valuable aid in obtaining modeling knowledge. As

implemented, the EMM power system model predicts nominal performance based on the commands which are sent to the spacecraft, enabling the EMM to detect anomalous behavior. This model does not constitute a device model as discussed previously, lacking among other things an explicit model of device interconnections. The model may be considered to contain only algorithmic knowledge of how the power system behavior may be predicted, and therefore need not even be considered part of the knowledge base, since the model could as easily be written in a lower-level procedural language. Such knowledge does not aid diagnostic reasoning, since it treats the power system as a "black box". Hence, the EMM is indeed restricted to shallow causal reasoning for the present.

The diagnostic knowledge was acquired primarily through interviews with JPL domain expert Dick Johnson. The knowledge acquired covered a representative class of faults involving event verification and anomaly interpretation. Not all of the knowledge has been implemented in the current rule base, since the goal of the prototype was to demonstrate the suitability of our chosen paradigm rather than to achieve broad coverage of the fault space.

Section 5: Conclusions and Recommendations

The prototype performed successfully on the various test scenarios, verifying events and detecting anomalies in a simulated telemetry stream. The prototype is successful in classifying faults which are represented in its knowledge base, moreover, it avoids a tendency to misclassify anomalies which it has not been prepared to diagnose, demonstrating the effectiveness of our diagnostic strategy. This success may be attributed to carefully written diagnostic rules, and also to the fact that the Galileo spacecraft's instruments were designed to provide information that would be complete and discriminating. Thus, we have good confidence that the diagnostics can be expanded to cover a broad space of operational faults in the Galileo PPS.

The power system model has demonstrated the ability to accurately predict Galileo telemetry for a broad range of scenarios. The model's chief shortcoming is the lack of knowledge to predict probable error tolerances in the telemetry channels. Other omissions include modeling of the RTG power output decay, the inverter efficiency curve, AC load power factors, and thermal variation in cabling resistance. These shortcomings can be easily rectified, and in any case do not compromise the functional goals for the prototype, since realistic scenarios are possible which do not violate the limitations of the current model. The power model is currently implemented in CLIPS rules, since this approach yields the best programming productivity in constructing the

prototype. The telemetry predicts may be computed far more efficiently in procedural code, which would be advantageous in view of the real-time constraints upon the system. Hence we recommend that this approach be taken in developing a fully implemented version of the EMM. Because the anomaly detection function is wholly dependent on the accuracy of the telemetry predicts, we would also recommend that the fully implemented power model be exhaustively verified against both the GPWR program and actual spacecraft telemetry.

It was concluded that the prototype does not adequately demonstrate event verification because the variability in time of arrival is not represented in the test cases. It is our view that event detection is not a proper task for a production system, nor is it efficient for the actual comparison of received telemetry against alarm limits to be implemented in production rules, since the overhead of the control algorithm is unnecessary and costly for these repetitive, procedural tasks. For these reasons, it is contemplated that a full-scale version of EMM would be coupled with an telemetry preprocessor which would apply an event detection algorithm to the telemetry stream, passing only the completed event records on to the production system for analysis. This change will have large impact on the knowledge base and functioning of the diagnostic system, and it is therefore a high priority development task.

The prototype only partially addresses the problem of real-time response. We have demonstrated that a quite spartan reasoning paradigm can be effective in Galileo PPS anomaly diagnosis, however, a true test of the prototype's throughput was not practical, due to the fact that Galileo telemetry is not yet available from SFOC. Thus, an important goal of the next development phase, full-scale demonstration, is to demonstrate the ability to monitor an actual Galileo telemetry stream in real time, while leaving adequate headroom for other processes which may be running on the host workstation.

Glossary

- "antecedent" The conditions necessary for drawing a conclusion. In a production system, the left hand side of a rule contains the antecedent conditions of that rule, while the right hand side contains the consequent.
- "backward chaining" A problem solving method which starts with a goal to be achieved and recursively expands each unsolved goal into a set of simpler subgoals until either a solution has been found or all goals have been expanded into their simplest components. when a subgoal is solved, it backs up its solution to its parent goal. In a production system, backward chaining starts from the consequent of the rule rather than the antecedent conditions.
- "behavior" is what the physical device actually does.
- "compiled knowledge" Knowledge that has been pre-processed into a form that is well suited for a particular fault diagnosis procedure. E.g., fault dictionary.
- "consequent" The conclusion of a rule or logical proposition.
- "coverage" is a measure of what proportion of the fault model the diagnosis is able identify.
- "deep reasoning" Reasoning from "first principles", i.e., from a circuit diagram and Ohm's law. More general than shallow reasoning.
- "demon" A rule which is fired upon its antecedent conditions being met. In an environment with backward-chaining control, demon rules are those which fire in forward-chaining discipline. In a forward-chaining system, demon rules may simply be those which have priority over all other rules, or which operate in all contexts.
- "design" A schematic representation of a physical device which satisfies the specification.
- "design verification" verifies that design's behavior satisfies the specification, similar to diagnosis.
- "device" An electronic or electromechanical system or component that is subject to fault diagnosis.
- "device verification" verifies device against the faults in the chosen fault model, similar to monitoring and diagnosis.
- "diagnostic task" attempts to identify the cause of a divergence within the chosen fault model.
- "expert system" A computer program that has expertise in a narrow domain. The expertise commonly results from problem-specific knowledge rather than generalized reasoning power.
- "fault dictionary" a two-dimensional table which lists the symptoms that result from each of a number of possible faults in a specific device.
- "fault model" is a subset of the faults which could occur, usually defined by assumptions such as 'the fault is a single-point failure' or 'the fault is non-intermittent'.
- "fault tree" a type of decision tree in which terminal nodes are fault diagnoses.
- "fault-based" an approach to fault diagnosis that is based on a known mapping from symptoms to causes, such as a fault dictionary or fault tree.

"forward chaining" A problem-solving method that starts with initial knowledge and applies inference rules to generate new knowledge utilizing either one of the inferences satisfies a goal, or no further inferences can be made.

"frame" An extremely general structure for knowledge representation, generally used in more powerful reasoning systems.

"goal" The end to which problem-solving aims.

"granularity" The scale of the primitive, or atomic, components of the design.

"heuristic" A principle (sometimes called a rule-of-thumb) that embodies some problem solving knowledge. A heuristic does not always guarantee success. Heuristics are commonly implemented through rules.

"hybrid reasoning" is a process that incorporates aspects of shallow reasoning and deep reasoning, in order to benefit from the speed of the former and the generality of the latter.

"inference chain" A tree structure showing causation from premises (leaf nodes) to a conclusion, the root node of the inference train.

"inference engine" The algorithm by which inference rules are applied to the knowledge base to derive conclusions. See "forward chaining" and "backward chaining".

"knowledge base" Those facts and inference rules which are domain-specific, as distinct from the inference engine.

"metaknowledge" Knowledge which the system may possess regarding its own domain-specific knowledge and how to apply it in problem-solving. Considered to be implicit in typical expert systems, research continues on ways to represent this knowledge explicitly in the knowledge base.

"model-based" An approach to fault diagnosis that is based on a model or simulation of a device. In principle, if all of the interconnections within the device are included in the model, the symptoms of any device behavior can be accounted for, as opposed to a fault-based diagnosis where only the faults and symptoms compiled in the the fault dictionary can be identified.

"production system" Alternate term for expert systems based on a forward-chaining inference engine.

"resolution" describes how specifically the cause of a divergence can be isolated. In the model-based diagnostic approach, this is influenced by the granularity of the device model.

"rule" A representation of the relationship between a situation and an action. Rules are ordered pairs with a left hand side (antecedent) and a right hand side (consequent).

"search space" The set of alternatives from which a solution to a problem is to be found. The set may be explicitly or implicitly defined.

"semantic network" A system for representing objects, their properties and their relationship to other objects, e.g., taxonomies.

"shallow reasoning" Explanation based on relationships that lead relatively directly from symptoms to causes. Superficial knowledge which is valid only in a certain context. More efficient than deep reasoning.

"specification" is a formal description of the desired device behavior.

"tool" A tool for constructing knowledge systems is intermediate between an implementation language such as LISP and a shell in terms of its power and flexibility. Avoids limitations of shells while enhancing productivity compared to LISP.