Intelligent Fault Isolation and Diagnosis for Communication Satellite Systems

Donald P. Tallo, John Durkin
The University of Akron
Akron, Ohio
and
Edward J. Petrik
NASA Lewis Research Center
Cleveland, Ohio

ABSTRACT

NASA-Lewis Research Center recently completed the design of a Ka-band satellite transponder system, as part of the Advanced Communication Technology Satellite (ACTS) System. To enhance the reliability of this satellite, NASA funded The University of Akron to explore the application of an expert system to provide this satellite with autonomous diagnosis capability. The result of this research was the development of a prototype diagnosis expert system, called FIDEX (Fault Isolation and Diagnosis EXpert).

FIDEX is a frame-based system that uses hierarchical structures to represent such items as the satellite's subsystems, components, sensors, and fault states. This overall frame architecture integrates these hierarchical structures into a lattice that provides a flexible representation scheme and facilitates system maintenance. To overcome limitations on the availability of sensor information, FIDEX uses an inexact reasoning technique based on the incrementally acquired evidence approach that was developed by Shortliffe during his MYCIN project. The system is also designed with a primitive learning ability through which it maintains a record of past diagnosis studies. This permits it to search first for those faults that are most likely to occur. And finally, FIDEX can detect abnormalities in the sensors that provide information on the transponder's performance. This ability is used to first rule out simple sensor malfunctions.

The overall design of the FIDEX system, with its generic structures and innovative features, makes it an applicable example for other types of diagnostic systems. This paper discusses these aspects of FIDEX, and illustrates how they can be applied to fault diagnostics in other types of space systems.

Key Words: Expert System, Space Systems, Communication Satellite Systems, FDIR Diagnostics, Frame-Based, Abstract Reasoning, Learning, Sparse Sensors, Sensor Validation

1.0 INTRODUCTION

The satellite network of the United States supports both the commercial and military sectors by providing an effective worldwide communication network. The reliability of this network represents a strategic resource for this country and a critical concern for the National Aeronautics and Space Administration (NASA). Since the mid 1980's, NASA has been investigating the application of expert system technology as a means for improving satellite reliability. The principle motivation for such work has been to develop an intelli-

gent expert system that could be placed onboard a satellite, permitting the satellite to perform autonomous diagnosis. Success in this effort would offer the potential of significantly improving the reliability of satellite communication systems.

In the summer of 1988, NASA-Lewis Research Center funded The University of Akron to study the application of such a diagnosis expert system.

1.1 Overview of Application Area

NASA has recently completed the design of a Ka-band (30/20-GHz) communication satellite transponder. This transponder system is to be integrated within the Advanced Communication Technology Satellite (ACTS) System and deployed early in 1993.

The ACTS transponder is a multiple channel repeater that relays microwave communication signals between highly localized ground terminals; see Figure 1.1. All references to the transponder in this paper are directed towards the components of the communication system that will reside onboard the satellite.

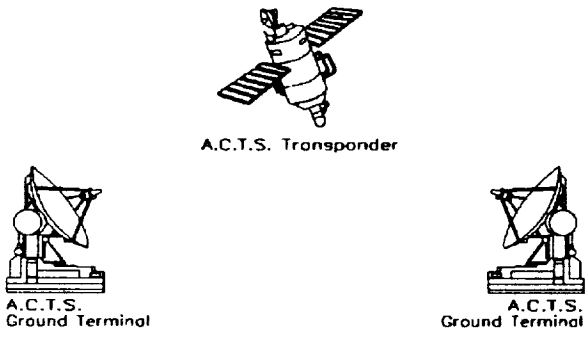

Figure 1.1 ACTS System

Figure 1.2 shows a schematic representation of the ACTS transponder. At present, only two of the multiple channels are implemented in its design. However, this proof of concept design can easily be expanded to incorporate additional links as the system design progresses.

At present, the design of this transponder is being evaluated within the System Integration, Test, and Evaluation (SITE) testbed at NASA-Lewis. The SITE laboratory is used by NASA for validating designs and demonstrating the capabilities of satellite communications systems. This phase of development is valuable to NASA for refining the response of the various systems onboard the transponder. Another important aspect of SITE is the formulation of an understanding of these systems' fault response.

1.2 Project Definition

The goal of this research project was to investigate the possibility of representing the knowledge gained during this SITE phase in a diagnostic expert system. Such a study would then help to lay groundwork for a future system capable of providing the transponder with autonomous diagnosis capability.

The research for this project progressed according to several key developmental phases:

1. *Domain Analysis*: Study the operation of the application system under both normal and abnormal conditions
2. *Knowledge Acquisition*: Study and organize the knowledge used by the domain experts who perform fault diagnostics on application system
3. *Knowledge Representation*: Design a scheme to model the application system and represent the knowledge required to detect, isolate, and diagnose its fault states
4. *Response Strategy Definition*: Establish response strategies and procedures for all fault states

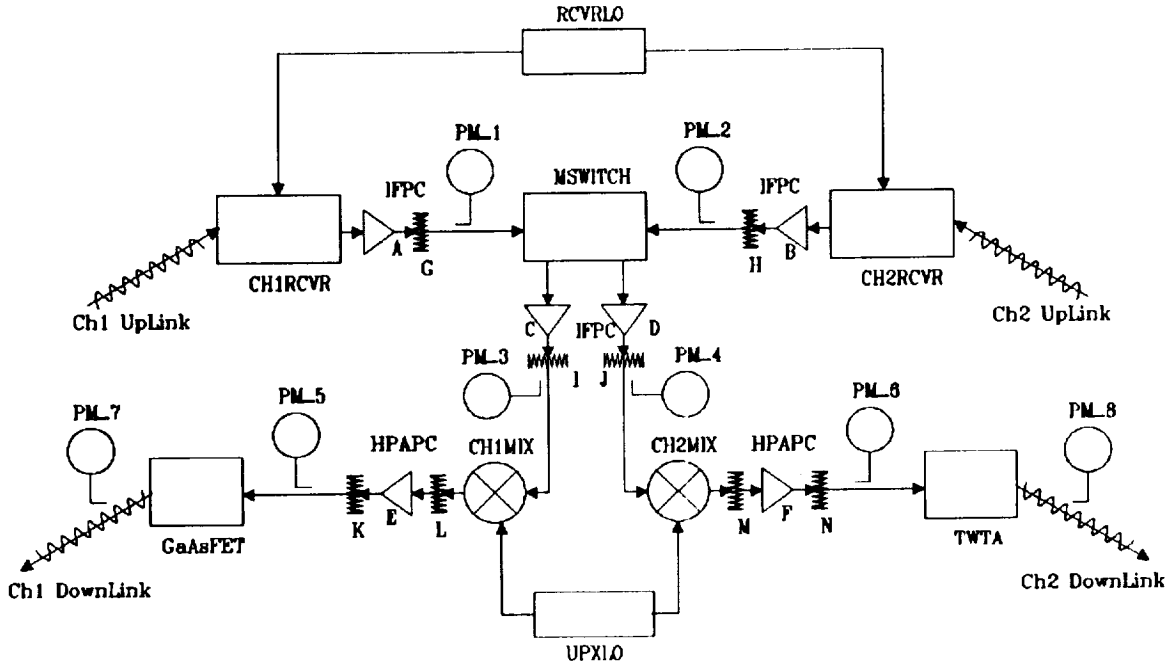

Figure 1.2 SITE Model of the ACTS Transponder System

5. *Prototype Development*: Develop, test, and modify a series of evolutionary diagnostic expert systems
6. *Requirements Definition*: Define the overall specifications for the final diagnostic expert system
7. *Final Development*: Design, encode, integrate, test, and document the deliverable expert system
8. *Life Cycle Analysis*: Define and specify a maintenance schedule for the deliverable diagnostic expert system

During these phases of development, several problems were encountered that reshaped the requirements of the project. Three problems of particular interest resulted from the evolutionary state of the ACTS transponder system. The requirements that these difficulties added to the project, and their solutions, highlight the major strengths of this expert system.

The first of these difficulties became evident during domain analysis. The expert system was constrained to work with limited information on the operational condition of the transponder. Specifically, there were only a few sensors available to provide information on the response of the transponder system. This information was limited to the signal power level sensors, indicated in Figure 1.2 as *PM_1* through *PM_8*, and a few bit error rate (BER) registers. This limited information was not completely adequate for assessing the condition of the transponder. In short, the sensors in the transponder were sparse in number, compared to the other components of the transponder system. Therefore, the isolation of a fault to a specific component based upon sensory information alone was not possible. This limitation was termed the *Sparse Sensor Problem*.

This problem also placed a high premium on the reliability of sensory information. Inconsistent or erroneous readings could render the expert system inoperable. Therefore, a method for resolving conflicts in sensory data was needed.

A second problem was encountered during knowledge acquisition. A prerequisite for the development of any expert system is an extensive understanding of the application area. In a diagnostic application, this requirement dictates that the potential fault states of the system be well known. However, the ACTS transponder was still under evaluation, and a complete understanding of its fault response had yet to be formulated. This fact constrained the investigators to work with limited diagnostic knowledge. Without a clear definition of the transponder's fault response, explicit diagnostic rules were not possible. Therefore, the expert system was prescribed to work with abstract, rather than concrete, diagnostic knowledge.

The final problem was also a result of the evolutionary state of the transponder system. The problem was that changes in the design of the system were always possible. These changes could range from modifications to design specifications, or even the addition of new modules. This situation made it difficult to develop a robust diagnostic agenda.

Faced with these problems, the goal of this project changed more towards a study effort. Emphasis was placed on the development of techniques that would overcome these problems and permit the expert system to reason intelligently with only limited information. The system's knowledge needed to be structured such that any change in the design of the transponder could easily be reflected in the structure of the expert system. All of these requirements placed a premium on the design of knowledge representation techniques and reasoning methods that were general and flexible. The result of this effort was the development of a prototype diagnostic expert system called FIDEX, Fault Isolation and Diagnosis EXpert. This project demonstrated the feasibility of developing an intelligent computer diagnostic system not only for the ACTS transponder, but for space systems in general.

1.3 General Approach to Solution

The general approach taken in the development of this project followed the problem-solving approach used by the ground personnel who perform satellite diagnostics. This strategy was termed the *Modular Approach to Diagnostics*. In general, it follows the four tasks defined below.

1. *Fault Detection*: Monitor the response of the transponder to determine whether it is functioning properly or not
2. *Fault Isolation*: Narrow the range of suspected components to the smallest possible group
3. *Fault Diagnosis*: Investigate the precise nature of the misbehavior and determine the component causing it
4. *Fault Response*: Respond to the diagnosis in a robust and intelligent manner

The purpose of the first task, *Fault Detection*, is to detect any misbehavior in the transponder performance. This task involves the analysis of current sensor information to ascribe qualitative descriptions to each sensor's reading; either *"GOOD"* or *"BAD."* These descriptions are based on whether the data reported by a sensor exceed a tolerance figure centered on its nominal or expected value. Sensor readings that are within tolerance receive a *"GOOD"* description, and those that exceed their tolerance range are labeled as *"BAD."* The detection of a fault is based upon establishing a *"BAD"* reading on any sensor. This indicates that a misbehavior exists in the transponder system and causes the next task to begin.

The second task in this approach is *Fault Isolation*. Its purpose is to isolate the suspected fault to the smallest possible group of components in the transponder. This is accomplished through a principle known as *Error Propagation*. This principle states that the observable symptoms of a misbehavior in a component will propagate through all subsequent sensors in a signal path. The source of such a misbehavior can thus be concluded to lie in that signal path, prior to the detection of the misbehavior, and subsequent to the last sensor indicating a proper signal response.

To implement this, the isolation task considers the qualitative description of all sensor readings as ascribed by the detection phase. It locates a sensor reporting a *"GOOD"* reading that is followed by a *"BAD"* reading. However, because of the sparse sensor limitations, this approach can only isolate the source of the misbehavior to the group of components between these two sensors. For the purposes of this project, these groups of components are termed *SubSystems*, and are defined as the groups of components bounded by signal power level sensors.

The fault isolation task relies heavily upon the integrity of the data reported by the sensors. Should any sensor report erroneous data, this task will fail to reach a valid conclusion. Therefore, a subordinate *Sensor Validation* task was added to this diagnostic phase.

The sub-task of sensor validation is designed to identify the possibility of a faulty sensor. This ability permits the FIDEX system to avoid the search for a non-existent transponder fault. Sensor validation is also based on error propagation; however, in a slightly different fashion. Again, a signal producing a *"BAD"* sensor reading at one point in the transponder should result in a *"BAD"* reading on all subsequent sensors in that signal path. This task identifies the possibility of a faulted sensor if a *"GOOD"* reading instead is found.

In either case, the purpose of isolation is to identify the subsystem containing the component causing the misbehavior. If this misbehavior is the result of a component failure, the subsystem identified by its input and output sensor readings is flagged as isolated. However, if the detected *"BAD"* sensor reading is the result of a faulty sensor, isolation flags the sensory components as the isolated subsystem. Once the source of the fault is isolated, the next task is initiated.

The third task, *Fault Diagnosis*, involves consulting a community of diagnostic expert systems. Each system is designed to address the problems of a specific subsystem within the transponder. Determining the appropriate diagnostic expert to be consulted is the final task of the isolation phase.

These specialized diagnostic systems use knowledge that is rule-based and backward chaining in nature. The hypotheses for these rules represent the potential faults in the isolated subsystem. The order in which they are placed on the agenda is based on the history of the fault states. Maintaining this history permits FIDEX to pursue the most likely problems first.

Each diagnostic system was also designed with an ability to perform inexact reasoning. This was done to overcome problems that resulted from limited

information about the transponder's performance. Such an ability was important in that the FIDEX system would often need to make a "guess" at the most likely fault state.

The inexact reasoning technique chosen for this project was based on the certainty theory given by Shortliffe (1975), with some modification by Durkin (1991). It relies upon establishing incremental measures of belief or disbelief in rule conclusions. These two factors are then used to establish an overall confidence when a conclusion is supported by multiple rules.

The final task is *Fault Response*. The present strategy for fault response is to provide recommendations for reconfiguring the components or sensors. Plans are to include the capability to reconsider fault diagnosis if the recommended action was ineffective. FIDEX would retain its past diagnosis, including recommendations, and reconsider the problem with information made available following the corrections to the transponder.

The remainder of this paper discusses the workings of the FIDEX system. It will demonstrate the techniques discussed above and, by example, show their application to other types of diagnostic systems.

KNOWLEDGE REPRESENTATION

The diagnostic knowledge of FIDEX is represented using both frame-based and rule-based techniques. This section discusses the structure of that hybrid framework. It also provides sample code segments describing the actual implementation in the syntax of NEXPERT Object, the software development tool used in the project.

2.0 FRAME NETWORK STRUCTURE

The expert system needed to be designed such that it would easily allow the incorporation of changes to the transponder. Therefore, it was decided that a frame-based approach for knowledge representation would be appropriate.

Frame hierarchies were developed to represent the transponder's components, subsystems, sensors, and fault states. These hierarchies were interconnected into a network to enrich the overall knowledge representation structure.

2.1 Structure of Components Class

A frame hierarchy was created to provide a clear and efficient representation of all components in the transponder. Figure 2.1 shows this structure called the *Components Class*. This figure illustrates a convention that will be maintained throughout in this paper. Circles represent class frames and triangles represent object frames. Lines indicate links between frames, with the arrows indicating the direction of inheritance.

The root node in Figure 2.1 is a circle indicating a class frame called *Components*. This class was created to represent the commonality between all components in the transponder. It is divided into several subclasses represented by the second level of class frames. Each of these subclasses describes the function of components in the transponder: amplifiers, attenuators, etc. The components are represented by object frames attached to these subclasses.

The code segment describes this structure. The first series of declarations defines the properties that are to be used. This is not

a complete listing. Only the properties of interest in this discussion are shown.

Properties were defined to describe physical characteristics about a component: its name, input/output components, etc. These properties are used by FIDEX to give a component a self awareness. Other properties provide functional information about the components: its input and output signal power levels, gain, nominal gain, etc.

The next definition creates a class frame called *COMPONENTS* in the object space of the expert system. It establishes links to several subclasses and defines which properties will be associated with this class. Each subclass inherits all properties associated with the *COMPONENTS* class. Any properties specific to a type of component can be defined at the subclass level. The definition for the *ATTENUATORS* class is included as an example of this. The *SETTING* property is used to describe the variable attenuation setting of the attenuators in the transponder system.

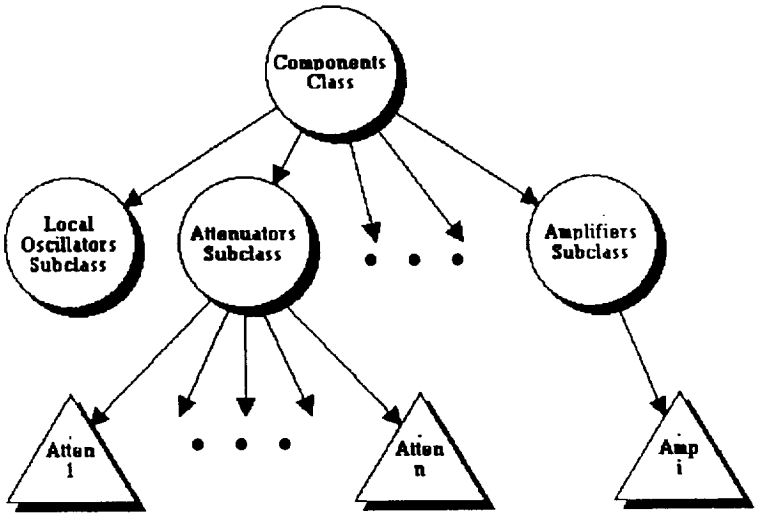

Figure 2.1 Components Class

```
(@PROPERTY = COMPONENTINN      @TYPE = String;)
(@PROPERTY = COMPONENTOUT      @TYPE = String;)
(@PROPERTY = FAILED            @TYPE = Boolean;)
(@PROPERTY = GAIN              @TYPE = Float;)
(@PROPERTY = GAINNOMINAL       @TYPE = Float;)
(@PROPERTY = NAME              @TYPE = String;)
(@PROPERTY = SETTING           @TYPE = Integer;)
(@PROPERTY = POWERINN          @TYPE = Float;)
(@PROPERTY = POWEROUT          @TYPE = Float;)

(@CLASS = COMPONENTS
        (@SUBCLASSES = AMPLIFIERS
                       ATTENUATORS
                       BERREGISTERS
                       GaAsFETS
                       LOCALOSCILLATORS
                       MIXERS
                       POWERMETERS
                       RECEIVERS
                       SWITCHES
                       TWTAS )
        (@PROPERTIES = COMPONENTINN
                       COMPONENTOUT
                       FAILED
                       GAIN
                       GAINNOMINAL
                       NAME
                       POWERINN
                       POWEROUT ) )

(@CLASS = ATTENUATORS
        (@PROPERTIES = SETTING ) )

(@OBJECT = IFPCATTEN1
        (@CLASSES = ATTENUATORS ) )
```

Finally, the last definition in the code segment shows the attachment of an object frame to this structure. An object frame called *IFPCATTEN1* is created in the object space to represent one of several *IF* signal *P*ower level *C*ontrol *ATTEN*uators in the transponder. This attenuator object is assigned to the *ATTENUATORS* class. Therefore, it inherits all properties assigned to both this class and the *COMPONENTS* class. Each component of the transponder is represented by an object frame in this manner.

2.2 Structure of Subsystems Class

Each component is also associated with a subsystem of the transponder (see Figure 2.2). Several object frames are used to represent the collections of components called subsystems. These frames are then organized by attaching them to a class frame for all subsystems in the transponder. Finally, the membership of a component in a particular subsystem is represented by attaching its object frame as a subobject of the appropriate subsystem object frame.

Again, a code segment is provided to describe this structure. Similar to the components definition, several properties are defined to represent both structural and functional information about the subsystems of the transponder.

A class frame called *SUBSYSTEMS* is created in the object space of the expert system. Properties assigned to this class are inherited by all attached object frames. Finally, the last definition in the code segment shows the assignment of an object frame to this structure.

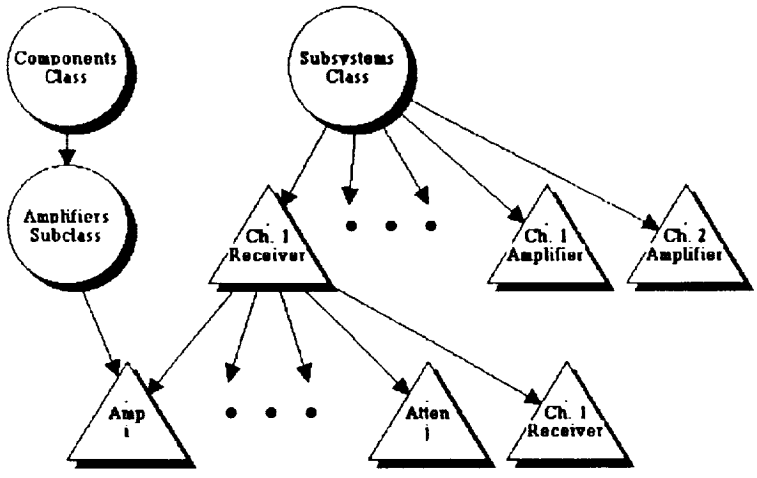

Figure 2.2 Subsystems Class

An object frame called *CH1RECEIVER-SYSTEM* is created in the object space to represent the group of components associated with the Channel 1 Receiver Subsystem. Object frames to represent the Channel 1 Receiver unit, an IF Signal Power Control Amplifier and Attenuator, and the Receiver Local Oscillator are attached as subobjects of this subsystem.

```
(@PROPERTY = ISOLATED          @TYPE = Boolean;)
(@PROPERTY = READINGINN        @TYPE = String;)
(@PROPERTY = READINGOUT        @TYPE = String;)
(@PROPERTY = SYBSYSTEMINN      @TYPE = String;)
(@PROPERTY = SUBSYSTEMOUT      @TYPE = String;)

(@CLASS = SUBSYSTEMS
          (@PROPERTIES = GAIN
                         GAINNOMINAL
                         ISOLATED
                         NAME
                         POWERINN
                         POWEROUT
                         READINGINN
                         READINGOUT
                         SUBSYSTEMINN
                         SUBSYSTEMOUT ) )

(@OBJECT = CH1RECEIVERSYSTEM
           (@CLASSES = SUBSYSTEMS )
           (@SUBOBJECTS = CH1RCVR
                          IFPCAMP1
                          IFPCATTEN1
                          RCVRLO ) )
```

As these frames represent components of the transponder, they are attached to the *COMPONENTS* class structure as well. This linking of component object frames to the components world can be interpreted as an *Is-A Link*. Links to the subsystems world represent *Part-Of Links*. That is, the IFPC Amplifier *Is A*n amplifier and is *Part Of* the Channel 1 Receiver system.

This approach not only aids the diagnostic tasks, but also provides an efficient coding approach. Through multiple inheritance, each subsystem component acquires information from two parents. One

provides information on performance while the other provides information on structure.

2.3 Structure of Sensors Class

Two types of sensory element monitor both the response of the transponder and the relayed signal. The first type is signal power level sensor. The other type represents the data stream bit error rate (BER) registers located within the ground terminal systems. The information used for diagnosis is provided by these sensors. These sensors were represented by creating the class structure for all sensory components shown in Figure 2.3.

This structure is divided into subclasses according to the two types of sensor. Each sensor is then represented by an object attached to the appropriate type subclass. The code segment creates this structure in the object space of the expert system.

Properties are defined to describe the *DATA* reported by a sensor, its *NOMINAL* value, the corresponding *ERROR*, and the *TOLERANCE* band of acceptable error magnitudes. A string property called *READING* is used for the qualitative descriptions that were introduced in section 1.3.

```
(@PROPERTY = DATA        @TYPE = Float;)
(@PROPERTY = ERRORR      @TYPE = Float;)
(@PROPERTY = NOMINAL     @TYPE = Float;)
(@PROPERTY = READING     @TYPE = String;)
(@PROPERTY = TOLERANCE   @TYPE = Float;)

(@CLASS = SENSORS
          (@SUBCLASSES = BERSENSORS
                         POWERSENSORS )
          (@PROPERTIES = DATA
                         ERRORR
                         NAME
                         NOMINAL
                         READING
                         TOLERANCE ) )
 (@CLASS = BERSENSORS
          (@SUBCLASSES = CH1BERSENSORS
                         CH2BERSENSORS ) )
 (@CLASS = COMPONENTS
          (@SUBCLASSES = BERREGISTERS
                         POWERMETERS ) )
 (@OBJECT = BER1
          (@CLASSES = BERREGISTERS
                      CH1BERSENSORS ) )
 (@OBJECT = PM1
          (@CLASSES = POWERMETERS
                      POWERSENSORS ) )
```

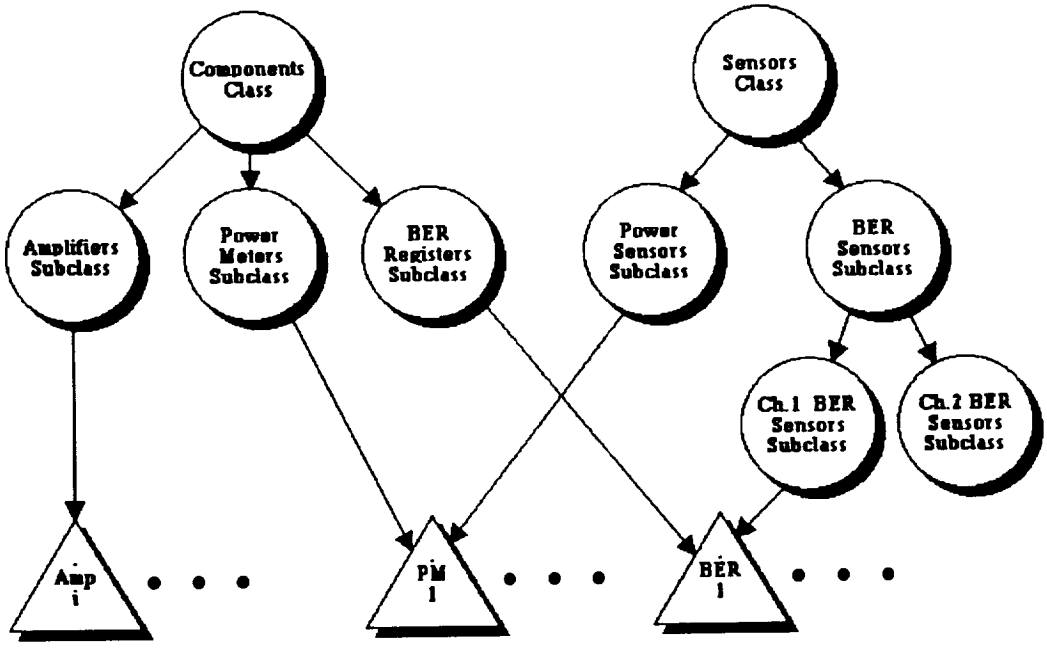

Figure 2.3 Sensors Class

The *BER SENSORS* class is also divided into two subclasses according to their channel. This was done to simplify the analysis of frequency-dependent fault states. It also demonstrates how class structures can be cascaded to further describe component function and organization.

Like all other transponder components, sensory elements could potentially fail. Therefore, each sensor is also represented in FIDEX as a member of the component world. The code segment shows the definition of two sensor type subclasses in the *COMPONENTS* world.

Each sensory component is represented by an object frame. The example shows the definition of one BER sensor, *BER1*, and one signal power level sensor, *PM1*. These frames are linked to their appropriate type subclass in both the components world and the sensors world.

2.4 Structure of Fault States Class

The transponder fault states are represented as objects in a class structure called Fault States. This class is also divided into several subclasses. Each subclass frame represents the association of fault states to component types: amplifier faults, attenuator faults, etc. Object frames representing the specific failure modes of the transponder are then attached to the appropriate subclasses. This structure, shown in Figure 2.4, enables FIDEX to reason about both known and abstract faults.

The code segment that defines this structure is nearly identical to that of the *COMPONENTS* class. This is because the type of fault state is associated with the type of component.

The primary properties associated with the *FAULT STATES* class are listed first. These describe which *COMPONENT* the fault is associated with, its *INFeRence CATEGORY* or priority, and the *POWER SYMPTOM GROUP* with which it is associated. A boolean property, *VERIFIED*, is used to flag fault states that have been verified by the diagnostic process. The final property listed is *VALUE*. This property is reserved by NEXPERT. The fault states represent the hypotheses of rules used during diagnosis. This property is assigned the results of rule evaluations.

The diagnostic process reasons with the fault state hypotheses using two distinct techniques. The next section discusses these, and provides structural information on their implementation.

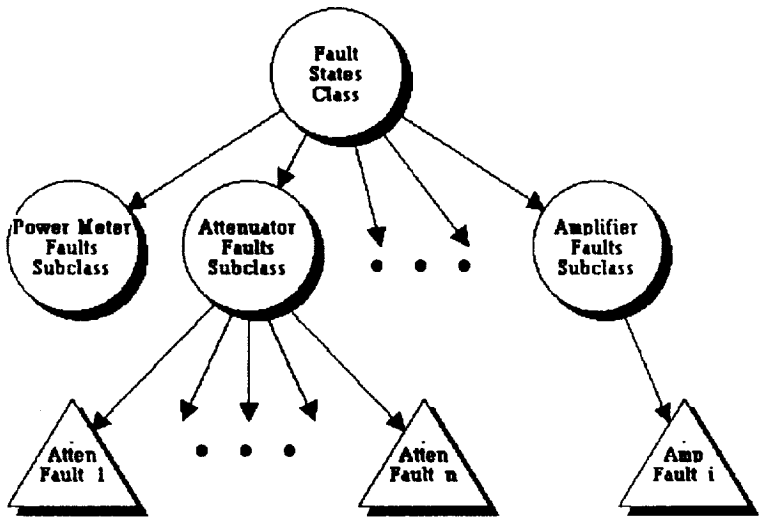

Figure 2.4 Fault States Class

```
(●PROPERTY = COMPONENT                ●TYPE = String;)
(●PROPERTY = INFRCATEGORY             ●TYPE = Integer;)
(●PROPERTY = POWERSYMPTOMGROUP        ●TYPE = String;)
(●PROPERTY = VERIFIED                 ●TYPE = Boolean;)
(●PROPERTY = Value                    ●TYPE = Boolean;)

(●CLASS = FAULTSTATES
        (●SUBCLASSES = AMPLIFIERFAULTS
                       ATTENUATORFAULTS
                       BERREGISTERFAULTS
                       GaAsFETFAULTS
                       LOCALOSCILLATORFAULTS
                       MIXERFAULTS
                       POWERMETERFAULTS
                       RECEIVERFAULTS
                       SWITCHEFAULTS
                       TWTAFAULTS )
        (●PROPERTIES = COMPONENT
                       INFRCATEGORY
                       NAME
                       POWERSYMPROMGROUP
                       VERIFIED
                       Value ) )

(●OBJECT = IFPCAMPSUPPLYFAILURE
        (●CLASSES = AMPLIFIERFAULTS ) )
```

3.0 REASONING TECHNIQUES

FIDEX reasons with two distinctly different techniques. The first technique, *Absolute Reasoning*, is used to establish or reject the existence of concrete, pre-defined fault states. The second technique, *Abstract Reasoning*, is used to recover when the diagnostic task cannot reason effectively using the first technique. Under such conditions, the second technique is used to establish evidence in conceptual fault states.

3.1 Absolute Reasoning

In general, knowledge that supports rules in absolute terms is *Associative Knowledge*. This type of knowledge associates conditions with the establishment or rejection of a conclusion. FIDEX uses two types of associative knowledge.

The first type is *Directly Associative*. This knowledge directly associates conditions with conclusions. An example of this type of knowledge might be: *If the data reported by a sensor reading exceeds its tolerance band, then the sensor's reading is "BAD."*

The condition of sensor data exceeding its acceptable range is directly associated with establishing a *"BAD"* qualitative description for that reading. Rules that represent this type of knowledge are used to structure the strategies of the diagnostic tasks.

However, the majority of the knowledge used in the task of fault diagnosis is supported by an accumulation of evidence. This type of knowledge is *Cumulatively Associative*. That is, the accumulation of several conditions is associated with the establishment or rejection of a conclusion. Moreover, each condition may contribute differently to that conclusion. An example of such knowledge might be: *A LOW signal power level might indicate internal phase lock failure in a local oscillator,* and *A HIGH bit error rate might indicate that the local oscillator is out of phase lock.*

Neither condition can be directly associated to establish or reject the conclusion of an internal phase lock failure. However, each contributes evidence to that conclusion. When multiple rules contribute evidence toward a conclusion, the system must be able to accumulate this evidence. The FIDEX system has such an ability.

3.2 Incremental Accumulation of Evidence

FIDEX uses the *Incremental Accumulation of Evidence* to establish or reject hypotheses that are supported by multiple rules. The technique used by FIDEX follows the work done by Shortliffe (1975) in his MYCIN project, with some modifications by Durkin (1991).

The first two equations given below accumulate a measure of belief, AB, and disbelief, AD, in a hypothesis, H. These

two measures are then used by the third equation to establish an overall confidence, *CF*, in that hypothesis. These equations work as follows.

$$AB(H)_k = AB(H)_{k-1} + MB(H)_k \cdot [\ 1 - AB(H)_{k-1}\ ]$$

$$AD(H)_k = AD(H)_{k-1} + MD(H)_k \cdot [\ 1 - AD(H)_{k-1}\ ]$$

$$CF(H)_k = [\ \frac{AB(H)_k - AD(H)_k}{1 - \min(AB(H)_k, AD(H)_k)}\ ]$$

Rules that accumulate knowledge do not assign boolean values to their associated conclusions. Instead, they determine a measure of belief, *MB*, or measure of disbelief, *MD*, in that conclusion. These measures represent the degree to which the conclusion of that rule has contributed to the establishment or rejection of its hypothesis. The values that are assigned to these measures range between 0 and 1. Values close to 1 represent strong measures while values close to 0 represent weak measures. A value of 1 is generally not assigned, as it results in a boolean value for *AB* or *AD*.

Consider an arbitrary hypothesis, *H*, and assume that no evidence has been established toward belief in that conclusion, $K = 0$ and $AB(H)_0 = 0$. Establishing a fact in support of this conclusion might assign a measure of 0.2 to the belief in *H*, $MB(H)_1 = 0.2$. The accumulated belief in the hypothesis would then be: $AB(H)_1 = 0.2$. The establishment of another piece of evidence in support of *H* might assign a measure of 0.5 to the belief in *H*, $MB(H)_2 = 0.5$. The accumulated belief in the hypothesis would then be incremented: $AB(H)_2 = 0.6$.

The accumulated measure of disbelief, *AD(H)*, is incremented similarly. However, this accumulation would be founded on rules that establish measures of disbelief in a hypothesis, $MD(H)_k$. This measure indicates evidence in opposition of the hypothesis.

As rules ascribe $MB(H)_k$'s and $MD(H)_k$'s, and accumulated values are calculated, the overall confidence in a conclusion, $CF(H)_k$, is calculated. Confidence factors range in value from -1 to 1. A value near -1 signifies little confidence in the conclusion, or the rejection of the hypothesis. A value near 1 denotes a high level of confidence, or the establishment of the hypothesis. Values in between represent various degrees of confidence, with 0 meaning unknown.

The following segments of code implement this technique. First, properties are defined to represent the *MB*, *MD*, *AB*, *AD*, and *CF* values required by the method. A string for a qualitative description of *CONFIDENCE* in a hypothesis is also defined.

```
(@PROPERTY = AB          @TYPE = Float;)
(@PROPERTY = AD          @TYPE = Float;)
(@PROPERTY = CF          @TYPE = Float;)
(@PROPERTY = CONFIDENCE  @TYPE = String;)
(@PROPERTY = MB          @TYPE = Float;)
(@PROPERTY = MD          @TYPE = Float;)

(@CLASS = CERTAINTYANALYSIS
          (@PROPERTIES = AB
                         AD
                         CF
                         CONFIDENCE
                         MB
                         MD ) )

(@OBJECT = IFPCAMPSUPPLYFAILURE
           (@CLASSES = AMPLIFIERFAULTS
                       CERTAINTYANALYSIS ) )
```

Next, a class for *CERTAINTY ANALYSIS* is defined and assigned these properties. All frame objects that require certainty analysis can then be attached to this class frame per the *IFPCAMP SUPPLY FAILURE* fault state shown in the example.

The primary purpose of this class structure is to provide the overhead required to

ascribe qualitative descriptions for *CONFIDENCE* in a hypothesis. The inference process for this assignment is triggered by active facets associated with these properties.

Methods are assigned to the *MB*, *MD*, *AB*, *AD*, and *CF* properties, and inherited by all object frames attached to this class. Three types of method are used. The first type, *Initial Value*, defines parameters to be used to initialize property values on reset or initialization of the inference process. The second type, *Order of Sources*, defines procedures to be taken to establish property values during the inference process. The final type, *Change Actions*, provides procedures to be followed when a property value changes.

In the syntax of NEXPERT, such methods associated with properties are called "meta-slots." The following code segment defines the meta-slots assigned to *CERTAINTY ANALYSIS* properties. They are inherited by all object frames that are attached to this class.

```
(●SLOT = CERTAINTYANALYSIS.AB
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do ((SELF.AB - SELF.AD) /
                         (1 - min(SELF.AB, SELF.AD))) (SELF.CF)))

(●SLOT = CERTAINTYANALYSIS.AD
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do ((SELF.AB - SELF.AD) /
                         (1 - min(SELF.AB, SELF.AD))) (SELF.CF)))

(●SLOT = CERTAINTYANALYSIS.CF
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Reset (Evaluate Certainty Factors))
                    (Do (Evaluate Certainty Factors)
                    (Evaluate Certainty Factors))))

(●SLOT = CERTAINTYANALYSIS.MB
       (●INITVAL = 0.0)
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do (SELF.AB + SELF.MB * (1 - SELF.AB))
                    (SELF.AB))
                    (Reset (SELF.MB))))

(●SLOT = CERTAINTYANALYSIS.MD
       (●INITVAL = 0.0))
       (●SOURCES = (RunTimeValue (0.0)))
       (●CACTIONS = (Do (SELF.AD + SELF.MD * (1 - SELF.AD))
                    (SELF.AD))
                    (Reset (SELF.MD))))
```

The initial conditions of the attached objects are assured by first setting these property values to 0. In that these objects represent hypotheses, this establishes the state of the system to $K = 0$ at both initialization and run-time.

When a value is assigned to the *MB* or *MD* property of an object in this class, the change actions inherited from these slots will fire. The equations for the accumulation of belief/disbelief are evaluated, and the result assigned to the current object's *AB* or *AD* property. *SELF* denotes the current object in class level definitions. The value of that property is then reset to its run-time value, 0.

With the assignment of a new value to either the *AB* or *AD* property, inherited change actions will fire. These actions evaluate the equation for the confidence factor and assign that value to the objects *CF* property. And, in turn, change actions associated with that slot fire. However, the actions taken by that slot are more complicated because it manipulates the agenda.

The NEXPERT agenda is a prioritized list of hypotheses to be pursued. When hypotheses are placed on the agenda by conventional means, they are pursued in an order defined by their priority, or inference category. However, this protocol can be overridden through the direct assignment of hypothesis to another value.

The purpose of the change actions associated with the *CF* slot is to affect the ascription of a qualitative description to the current object's *CONFIDENCE* property. This requires the evaluation of nine rules corresponding to as many delineations. Each rule has the same hypothesis, *Evaluate Certainty Factors*.

First, the change actions reset the value of this hypothesis to unknown. Then the value of the hypothesis is assigned to itself. Since the value of the hypothesis in unknown, this assignment of the hypothesis to itself forces the evaluation of its supporting rules.

Three of the nine rules that support this hypothesis are given in the following sample code segment. The example demonstrates the ascription of qualitative descriptions for *CF* values near 1 and -1, and an arbitrary range in between.

```
(●RULE = RULE 23 EVALUATION OF CERTAINTY FACTORS
      (●LHS = ( ≥ ( \CURRENT FAULT.NAME\.CF ) ( 0.9 ) ) )
      (●HYPO = Evaluate Certainty Factors )
      (●RHS = ( Let ( \CURRENT FAULT.NAME\.CONFIDENCE )
                  ( "ESTABLISHED" ) )
            ( Let ( \CURRENT FAULT.NAME\.VERIFIED )
                  ( TRUE ) ) ) )

(●RULE = RULE 24 EVALUATION OF CERTAINTY FACTORS
      (●LHS = ( ≤ ( \CURRENT FAULT.NAME\.CF ) ( -0.9 ) ) )
      (●HYPO = Evaluate Certainty Factors )
      (●RHS = ( Let ( \CURRENT FAULT.NAME\.CONFIDENCE )
                  ( "REJECTED" ) )
            ( Let ( \CURRENT FAULT.NAME\.VERIFIED )
                  ( FALSE ) ) ) )

(●RULE = RULE 27 EVALUATION OF CERTAINTY FACTORS
      (●LHS = ( ≥ ( \CURRENT FAULT.NAME\.CF ) ( 0.5 ) )
            ( < ( \CURRENT FAULT.NAME\.CF ) ( 0.75 ) ) )
      (●HYPO = Evaluate Certainty Factors )
      (●RHS = ( Let ( \CURRENT FAULT.NAME\.CONFIDENCE )
                  ( "POSSIBLE" ) ) ) )
```

The conditions, *@LHS*, of *RULE 23* will be true if the value of the *\CURRENT FAULT.NAME*'s *CF* is greater than or equal to 0.9. *CURRENT FAULT* is a blackboard object in FIDEX. Prior to the assignment of any certainty analysis measures, the name of the current fault state is posted to the *NAME* property of this object. This generic rule looks to the blackboard to determine the name of the current fault. Its *CF* is tested and the hypothesis accordingly established or rejected.

If the hypothesis is established, several actions are taken, *@RHS*. These actions first assign a qualitative description of *"ESTABLISHED"* to the *CONFIDENCE* property of the current fault. Then its *VERIFIED* property is set to *TRUE*. These rules are non-exhaustive. Therefore, the firing of one rule will terminate the evaluation of certainty factors.

RULE 24 evaluates a *CF* value at the opposite end of the scale. If this value is less than or equal to -0.9, the confidence in the hypothesis is described as *"REJECTED"* and its verification property is set to *FALSE*.

The final rule, *RULE 27* given as an example, shows the evaluation of a certainty factor in a range between those bounds. This rule states that if the *CF* of the current object is less than or equal 0.75 and greater than 0.5, the confidence in the current fault state is *"POSSIBLE."* This level of confidence is not sufficient to establish or reject the fault. Therefore, no assignment is made to its verification property.

Discussion to this point has been on the incremental accumulation of evidence toward concrete fault states. The next topic will discuss the application of these techniques for abstract reasoning.

3.3 Abstract Reasoning

In general, knowledge that supports rules in abstract terms is *Conceptual Knowledge*. This type of knowledge is *Indirectly Associative Knowledge*. It associates condition to abstract ideas that are indirectly related to the rule being pursued. An example of this type of knowledge might be: *A HIGH bit error rate is typical of a misbehavior in one of the frequency conversion components.*

FIDEX uses this type of reasoning to establish levels of confidence in class level

fault categories. That is, it might reach a conclusion of the form: *The observed symptoms are typical of those associated with a failure of the local oscillator.*

During the diagnostic task, FIDEX exhausts its knowledge about the fault states of the system. It is entirely possible that a failure mode might occur for which FIDEX has no knowledge. In that case, it would resort to confidence accumulated in class level fault states as its diagnostic conclusion.

This abstract reasoning ability of FIDEX is implemented as follows. First, all of the fault state type subclasses defined in section 2.4 are attached as subclasses of the class *CERTAINTY ANALYSIS*. Therefore, they inherit this class overhead.

By doing this, measures of belief and disbelief can also be assigned to the class properties. Levels of confidence can then be accumulated at this class, or conceptual, level. An example of such an assignment is given in the following rule.

This is a directly associative rule that establishes a qualitative description for the bit error rates during diagnostics. One of several hypotheses that are indirectly associated with concluding a *"HIGH"* bit error rate is that this symptom is associated with the class of *LOCAL OSCILLATOR FAULTS*. Therefore, the last two actions of this rule assign a measure of 0.3 to the belief that the fault is associated with a local oscillator.

```
(@RULE = RULE47 HIGH BIT ERRORR RATE
        (@LHS = ( ≤ ( <|CH1BERSENSORS|>.RATE ) ( 0.01 ) ) )
        (@HYPO = BitErrorrRatesAreHIGH )
        (@RHS = ( Let ( <|CH1BERSENSORS|>.RATE ) (/HIGH" ) )
           (Let (CURRENTFAULT.NAME) (// |LOCALOSCILLATORFAULTS|"))
           ( Do ( 0.3 ) ( \CURRENTFAULT.NAME\MB ) ) ) )

  (@SLOT = IFPCAMPSUPPLYFAILURE
           @INFATOM = IFPCAMPSUPPLYFAILURE.INFR CATEGORY; )
```

Using this technique, FIDEX can piece together information and reach conceptual conclusions such as the one given above. The final topic in this section is the representation of FIDEX's learning capacity.

4.0 LEARNING & SEARCH STRATEGY

There are two databases used by FIDEX. One contains information required to initialize parametric values of the system. Each record contains information on nominal readings, error tolerances, and other initial parameters. These values are loaded and stored in the appropriate slots of objects at runtime or when FIDEX is initialized. This method of initialization was chosen to facilitate the maintenance of the system.

The second database is used to provide FIDEX a limited learning capability. FIDEX stores the failure history of the transponder system in this database. Each known fault state is represented by a record that contains fields that represent the failure history of that fault state. Following diagnostics, FIDEX increments the history of the identified fault. This record keeping is used to direct the search strategy of future sessions toward the most likely faults.

The search strategy is adaptive in that the priorities by which known fault states are placed on the agenda is based upon the values maintained in the history database. A class level property of all fault states is the integer *INFR CATEGORY*. The value of this property is retrieved from the database when the diagnostic task is initialized. This property is then assigned to the inference priority of the fault state hypothesis by slot actions. The previous example shows

such a slot for one fault state. All fault state inference atoms are similarly initialized.

When the diagnostic task establishes a known fault state, the value of its inference category is incremented accordingly. The updated value is then stored in the learning database.

5.0 SUMMARY

The prototype FIDEX system is the result of a study effort by The University of Akron, funded by NASA-Lewis Research Center. Its purpose was to demonstrate that expert system technology can be applied to enhance the reliability of satellite communication systems, in particular, the Ka-band Advanced Communication Technology Satellite Transponder.

The initial goal of this research was to develop and expert system to provide this satellite with autonomous diagnosis capability. As limitations prevented the autonomy of FIDEX, the project became more of a study effort. Its goal changed towards the development of techniques to overcome several limiting problems.

The resulting system used hierarchical frame-based structures to represent the structure and operation of the satellite. Other strengths of FIDEX included its use of inexact reasoning techniques, its primitive learning ability, and its capacity for detecting abnormalities in sensors.

The overall design of the FIDEX system made it an applicable example for other types of diagnostic system. This paper discussed these aspects of FIDEX, and illustrated how they could be applied to fault diagnostics in other types of space systems.

REFERENCES

Durkin, J, Tallo, D.P., Petrik, E.J., *"FIDEX: An Expert System for Satellite Diagnostics,"* Space Communications Technology Conference Onboard Processing and Switching, NASA Conference Publication 3132, Cleveland, Ohio, November 12-14,(1991)

Harmon, P., Maus, R., Morrissey, W., Expert Systems Tools & Applications , John Wiley & Sons, New York, New York (1988)

Kerczewski, R.J., Fujikawa, G., *"Performance Measurements for a Laboratory-Simulated 30/20 GHz Communication Satellite Transponder,"* 13th AIAA International Communication Satellite Conference, March (1990)

Pfleeger, S.H., Software Engineering: The Production of Quality Software, Macmillan, New York, New York (1987)

Shortlife, H.H., Buchanan, B.G., *"A Model of Inexact Reasoning in Medicine,"* Mathematical Biosciences, Vol.23 (1975)

Waterman, D.A., A Guide to Expert Systems, Addison-Wesley, Reading, Massachusetts (1986)