A98-31685

ICAS-98-6,6,2

DIAGNOSTIC FROM SYSTEM MODELS: THE ADAM EXPERT SYSTEM APPROACH

Ermanno Girardelli, Fabrizio Didò
Alenia Aerospazio, Aeronautics Division
Corso Marche 41, 10146 Torino - Italy
Email: Girard@liat.alenia.polito.it

Abstract

The ADAM (Aircraft Diagnostic And Maintenance) project is designed to build a support system for the unscheduled maintenance of mechanical and electronic systems. The problem faced is how to help technicians to fix the detected malfunctions in the shortest time. The support system proposes the best test sequence to identify the failure in the quickest way, avoiding unnecessary part replacement.

The aim of this paper is to describe the second version of ADAM. In the first release of the system the diagnostic process was represented by means of trees, while in the second release ADAM makes use of system models to avoid any necessity of knowing the troubleshooting procedures in advance.

ADAM has been conceived as a shell to develop diagnostic systems for several kinds of complex equipment, and this paper provides a practical example of how it applies to a typical fire protection system for a military aircraft. The paper will highlight the main differences compared to the traditional diagnostic approach and the possibility of influencing the system design during its development phase.

Introduction to aircraft maintenance

Before of describing the ADAM expert system we will briefly describe the evolution of aircraft maintenance.

There are two major constraints driving aircraft programmes: aircraft mission availability and support cost. Evolution of the maintenance concept has tended to reflect the need to optimize these two drivers.

Traditionally, "hard time" maintenance was the leading concept. Equipment was removed at specified periods of time, considering a fixed scheduling time, in order to allow an in-depth investigation of their status. This was when no or very few electronic self-test functions were installed on the aircraft, and the state of the art did not allow in-depth control functions to obtain significant maintenance data.

The resulting high support costs and relatively low aircraft availability suggested a requirement to monitor the status of equipment directly during the time it was operating on the aircraft. This could be achieved by means of probes, continuity checks, dedicated tests (built-in-test upon request) and software for the correlation of such data/parameters gathered from the various sources.

In such an environment, maintenance actions were performed applying the "on condition" maintenance concept, i.e. when the monitor function indicated that the piece of equipment was degrading (from the operational point of view).

With the adoption of safety analysis, further progress in maintenance programmes then became practical. By using redundancy and fail-safe criteria it was possible to accept that certain failures could occur during the flight, and to allow for their occurrence instead of trying to prevent them.

Naturally, this approach is effective only if the maintainer is able to notice the occurrence of a failure, in order to devise the appropriate maintenance actions when needed. This approach is called "condition monitoring".

Further evolution in maintenance programmes became possible as the "testability" concept matured. Early troubleshooting was usually based on the experience of mechanics who made use of available documentation, professional skills, some statistics and, to a certain extent, trial and error. Extensive use was made of specialized test equipment (when available) to isolate the aircraft zone or component where the failure had occurred. While resolving some of the problems, heavy reliance on specialized test equipment was frequently

21st ICAS Congress
13-18 September 1998
Melbourne, Australia

found to negatively impact on the time to repair and the cost of the maintenance action. It also has a knock-on effect on logistics in terms of the need to supply, maintain and deploy of such specialized test equipment.

The real driver for testability was the need to reduce the "no fault found" condition (i.e. to minimize the unnecessary removal of non-defective items), a penalizing influence on aircraft availability and support cost. The "no fault found" rate in the early years of most aircraft programmes is about 40-50% of the total fault arisings, with consequent impact on the support cost due to the high number of spares needed and the additional diagnostic time to localize and repair the true fault.

Today, we believe that the best way to decrease the "no fault found" condition is to monitor the functional status through the implementation of a testability programme during the design phase. This implies, of course, that if we want to know the true status of the aircraft, in order to remove only faulty items, it is necessary to introduce appropriate monitoring devices and test points in airborne systems and equipment during the design phase.

Microswitches, temperature and pressure probes, electrical integrity checkers etc. installed in the appropriate items of each system make it possible to monitor the operating parameters of the aircraft. Dedicated test software, loaded or resident in control units (e.g. computers which control and manage aircraft functionality) analyze such information and make the aircraft health status available to the maintenance crew as an output. The maintenance engineer is then able to drive the aircraft maintenance using the health status information.

This is quite a significant breakthrough, but has shortcomings in terms of maintenance, related to the methods of retrieval and display of the health information so gathered. Initially, each control unit stored the maintenance data in its own (non volatile) memory, requiring such information to be downloaded at equipment level by the user via an external hardware and usually needed software support to translate the memory store content into an input suitable for troubleshooting purposes.

Moreover, the diagnostic procedures were essentially "paper based" even if the starting point for the troubleshooting was provided by the aircraft itself, either system by system or item by item, and troubleshooting techniques were generally based on the use of external test sets supported by the practical experience of maintenance engineers.

As computing power grew, further evolution became possible with the introduction of new electronic checks and on-board monitoring to improve the performance of control units, with the enhanced processing capabilities being used to perform greater depth tests at subsystem/item level. Even considering the relative slowness imposed by airborne qualification procedures, it would be fair to say that on-board software, the number of digital links and airborne computing power are doubling every three years or less.

Probes are now installed in mechanical and electro-mechanical systems in order to control all the operating parameters and to generate the relevant maintenance data. For avionic systems (communication, navigation,. etc.) electronic functions can check each item to reduce the ambiguity of fault indications.

The need for monitoring a high number of signals and to process an increased number of parameters brought centralization of the maintenance data. On-board monitoring systems were developed in an extensive way in order to achieve two main goals: collecting all the in-flight BITE data in a single place, and accessing such data in the most effective way.

Electronic control units within aircraft systems continuously collect health data during aircraft operations. Such data is sent (through appropriate buses such as MIL-1553, ARINC 429) to the central electronic unit where they are stored and processed for subsequent retrieval. An example of such data analysis is the identification of the possible faulty items.

The second goal is related to the on-ground utilization where BITE data is accessed via various media, on and off-board, e.g. displays, printers, CDROMs, databus retrieval via external sources, etc.

It is now possible to place greater reliance upon the aircraft internal capabilities, in terms of error message recording, as the starting point for the troubleshooting process. There is hardly any need to refer to an "expert mechanic" in order to "manipulate" information gleaned from sensors in aircraft systems.

The user/maintainer can interface directly with the machine which is now able to provide reasonably

accurate failure indications. Some types of failure are, of course, beyond the capacity of present-day BITE to identify, and require the specialized skills of the trained maintenance engineer to resolve.

He (or she) will still have to refer to external sources such as the troubleshooting procedures for information. Traditionally, such procedures are bulky manuals intended to support the "diagnoser", and contain both the logical reasoning for the "fault finding" and information for access/remove/install/align/test operations. The content, size and weight of such documents tends to have given them a somewhat "user-hostile" profile, since apart from being heavy and bulky they are full of arcane cross-references to information in other documents and limit their information to the "single fault" condition. From the diagnoser's point of view, the multiple fault is usually the one that gives most difficulty to resolve.

The current trend is towards storage of all relevant procedures on a computer in order to improve information availability and reduce user workload. A portable computer can be used to directly access the aircraft system/device where the maintenance data has been gathered and stored, and such data will be used to trigger the computer program that generates troubleshooting procedures, which can be linked to any referenced procedures such as remove, install and test of components.

This is indeed a significant improvement in terms of logistic support but some aspects of such an approach still need to be improved:

- firstly, this process needs to initiate sufficiently far upstream in the design process to fully consider the diagnostic needs. It is common knowledge that the design and the maintenance analyses (FMECA, Fault Catalogue), once frozen, form the basis for the troubleshooting procedures. In the case of critical areas (for instance when the item with the highest failure probability has been found to require the longest time for removal and testing) modifying the design becomes difficult, because at that time in the design cycle it will probably be considered either too late or too expensive to implement.

- Secondly, there is the need for some practical means of collecting the experience gained by maintainers and diffusing it among all the users. Up to now this is achieved somewhat piecemeal by means of user feedback leading to document modifications which take a certain amount of time to implement.

Approaches to diagnostic support systems

Diagnostics can be seen as a process that starts from a set of symptoms and leads to the identification of the causes of malfunctions through a series of steps aimed at search space reduction. Two kinds of knowledge are involved: technical knowledge, coming from technical manuals (e.g. fault isolation) or design documentation and empirical knowledge, coming from the expertise of personnel.

Technical knowledge, i.e. information about the behaviour of an aircraft system, is readily available and well documented, while empirical or heuristic knowledge, which is the result of years of practical fault finding, tends to be widely but thinly spread among many people and so is difficult to retrieve. It also tends to be inconsistent from location to location for the same reasons.

The first approach that we pursued in building a diagnostic system was based on collecting knowledge about how to perform troubleshooting on a physical system, i.e. modeling how people solve the problem of identifying a failure starting from a given situation. While this works, we found that there are a number of drawbacks in such an approach. The first is that collecting such knowledge is very time-consuming; the second is that it is difficult to keep the knowledge up-to-date with respect to the evolving configurations, and to keep pace with the various system modifications. There is also the human factor to consider - potentially, each individual will try to solve the same problem in a different way. If you build a troubleshooting support system based on the experience of a single person you will save time but very likely the system will not be completely accepted by other people. It is quite difficult to merge the various approaches to the same diagnosis to minimize this aspect.

Another way to approach the problem of building troubleshooting support systems has risen from the shortcomings that we have just pointed out: instead of focusing our attention on modeling the behaviour of the maintainer, attention is shifted to the behaviour of the physical system under consideration, in order to infer the entire troubleshooting process from scratch, by using a model of the system under test.

Alenia has been involved for many years in this area of research. Initially our efforts were devoted to the implementation of heuristic support systems, as previously mentioned, while now our attention is moving towards model-based systems. The task is challenging because it is not at all simple to grasp all the relevant aspects of physical system behaviour.

The remainder of this paper will describe using an example the kind of research we have done in Alenia, highlighting the problems that were encountered, the solutions adopted and the limitations to the scope of our investigation.

A simple fire protection system

Fire protection systems are commonly used on aircraft as a means to inform the pilot, or the groundcrew of any fire or overheat occurrence. They are usually installed in the most critical points of the aircraft where fire could potentially occur and cause damage the aircraft or the personnel (e.g. engine bay, APU bay, cargo bay). They normally consist of detectors (electrical or pneumatic) which will inform the cockpit and all the dedicated computers of a fire/overheat occurrence.

We will describe a simplified example of a fire protection system. A single pneumatic detector will report the fire/overheat status to the cockpit and to the computer.

The description of the system is as follows (see figure 1):

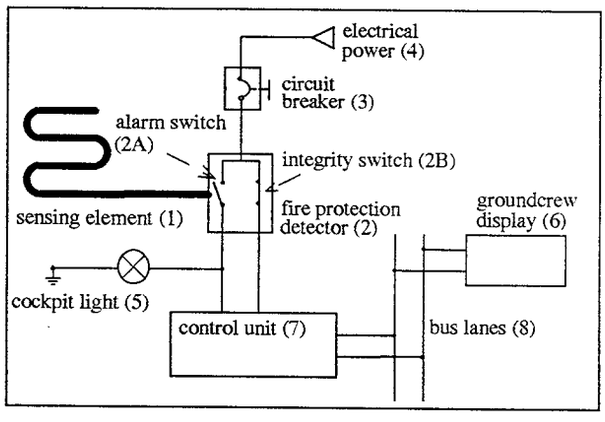

FIGURE 1 - Fire Protection Schematic Diagram

A sensing element (1) is installed throughout the zone which needs to be protected. On military aircraft such a zone is normally limited to the engine(s) and APU bay, whilst on civil aircraft it also includes the passenger zone and possibly other bays.

The sensing element is a thin metal tube filled with gas at a certain pressure, and when a fire/overheat event occurs the gas expands, forcing the alarm switch (2A) (which is fitted in the fire protection detector 2) to move, closing the circuit.

Then, the electrical power (4) will pass through the fire protection detector and two warnings will occur:

- the cockpit light (5) will inform the pilot that a fire/overheat event is occurring;
- the groundcrew display (6) will store and display the relevant warning for maintenance purposes. This will be driven by the control unit (7), which, once informed of the fire/overheat event via the signal from the detector (2), will send an appropriate bus message to the groundcrew display (6) through the bus lanes (8).

This is the active part of the system, i.e. how a fire/overheat hazardous event is displayed/advised to the pilot and to the maintenance groundcrew. The system is also provided with a passive self-monitoring capability.

In order to continuously monitor the condition of the sensing element, an integrity switch (2B) has been included. In the event of gas leakage from the sensing element, the internal pressure will fall, causing the integrity switch (2B) to open and the control unit (7) to sense the change of electrical status from closed to open.

The control unit will then supply the groundcrew display (6) with the appropriate error message through the bus lanes (8).

In order to protect the system from short circuits or over-current events, a circuit breaker (3) is placed upstream of the whole system architecture.

An example of "deep modeling"

We will describe here a "deep model" for the fire protection system described in the previous chapter. The last step will be to give a brief description of the algorithm that is able to infer a troubleshooting procedure from a deep model of a physical system.

The first steps when modeling a system is to describe those the entities relevant to the diagnostic process and to detail how such entities are connected. We have not yet developed a direct method for this, but in our experience the best way to choose such entities is to consider how maintenance operations are performed, and to describe only those items of equipment that will be subject to replacement or repair. This means that how you will develop a deep model of a system depends on how you intend to perform the maintenance on the system itself. For instance, when you allocate a failure to a computer, the normal diagnostic procedure can be either to replace the whole computer or to go in deeper detail with the troubleshooting to identify the faulty board. In the first case it would be enough to describe the computer with a single entity, in the second one you will probably need to use many entities to describe all the replaceable computer boards and devices. Initially you should assume that all possible recovery actions are codifiable and that they are reasonably stable over time, thus the choice of the entities to be described depends mainly on the location at which the troubleshooting is performed, which in turn identifies the probable user and therefore the kind of troubleshooting to be performed.

In this example we will assume that only simple replacement actions will be performed and that there will never be recovery actions concerning the internal components of a piece of equipment.

One aspect which immediately needs resolution involves the system boundaries, which means deciding where your system starts and ends, and how to consider those entities that actually do not belong to the system under consideration but which, nevertheless, are important for the completeness of the description. In our example we considered some indications on the groundcrew display, a device that very likely will not belong to the fire protection system as such but, for instance, may belong to a master warning system that lies outside the scope of our description. The solution to this kind of problem is a compromise between an exhaustive approach that would entail describing more than needed, and the correctness of the description itself. The above mentioned indications could be, for instance, directly connected to the computer itself, instead of describing the various pieces of hardware between the computer and the groundcrew display.

So, in our fire protection example, we could identify the following main entities:

control unit, electrical power, circuit breaker, ground, wiring, alarm switch, integrity switch.

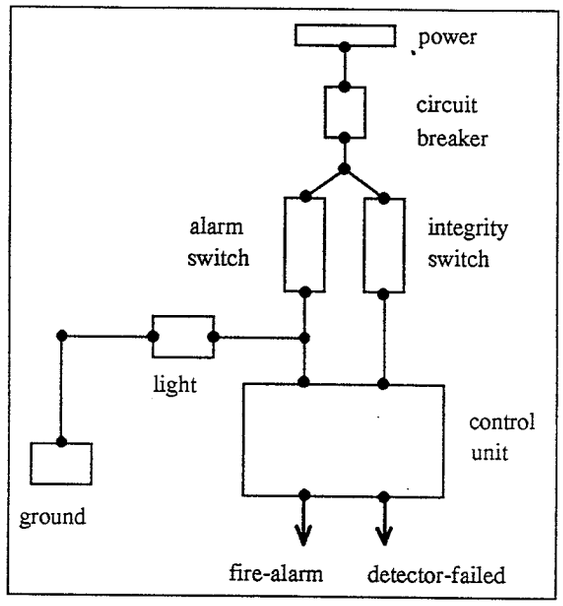

FIGURE 2 - Entities Diagram

A second step in the system modeling is the collection of standard information on each entity: the defect rate, the replacement time and the number of pins, where we use the term "pin" for addressing a channel of communication, of any nature, with the external "world".

The next step is then to analyze and describe the behaviour of each entity. This means first making a list of the all possible working and failure states. For instance we can describe a bulb as being a device that can be either in the "working" state or in the "failed" state, while a wire will normally be "closed" or "open" respectively when in the working or failure state. A circuit breaker has more possible states, the working conditions will be, for instance, "open" and "closed", while the failed states will be "failed-open" and "failed-closed", to mean a stable situation where the device cannot be placed in the closed status or the open one at will.

The last step in our deep-modeling process is to describe the behaviour of such entity by means of a formal language that makes use of clauses. A clause is something similar to a rule in rule-based expert systems, and at a first glance it is similar to a common "if-then" expression.

We are not going to describe the details of the formal language in this context, because we think that it is more useful to comment an example, trying to

highlight which is the right perspective to assume when describing the behaviour of a system.

Let us describe a simple circuit breaker (CB), a two pin device (that we will identify as pin1 and pin2 hereafter) that transfers electrical power only when in the closed status.

The clauses describing the circuit breaker are listed below:

Clause 1
Conditions: (Status 1) = Closed,
(Ampere pin1) <= 5
Functions: (Ampere pin2) = (Ampere pin1),
(Volt pin2) = (Volt pin1)

Clause 2
Conditions: (Status 1) = Open,
(Ampere pin2) = 0
Functions: (Ampere pin1) = 0,
(Volt pin1) >= 0

Clause 3
Conditions: (Status 1) = Closed,
(Ampere pin1) > 5,
(Volt pin1) >= 0
Functions: (Status 1) = Open,
(Ampere pin2) = 0

Clause 4
Conditions: (Status 1) = BlockedClosed
Functions: (Ampere pin2) = (Ampere pin1),
(Volt pin2) = (Volt pin1)

Clause 1 says that if the CB is "closed" and the electrical power is less that 5 Amperes then the device will remain in its original state, and all the electrical power will be transferred from pin1 to pin2 (another symmetrical clause describes the inverse flow).

Clause 2 says that when the CB is "open" the current will be zero on both pins, independent of the voltage on pin1.

Clause 3 describes the status change of the CB when the current is more than 5 Amperes, and the device goes from the closed status to the open one, with a reduction of the current to zero.

Finally clause 4 describes the only failure situation considered here, i.e. the CB stuck in the closed state and will not open, no matter how much current flows through it.

The clause language depicted here is powerful enough to describe many devices, but, at the same time has some limitations. For instance it is not possible to use it for describing complicated time-varying behaviour. One of the future steps in this research will be to improve the formal language in this direction.

What "modeling the behaviour" means

At this point we have given an idea on how a system can be modeled by means of entities and clauses. We described how to identify an entity, but we still have to identify those criteria that will drive us when modeling each entity. All these problems can be summed up in one question "Which are the aspects of the device that I need to model and which not?". When describing a transistor do I need to give a full description of all the quantitative aspects, by means of transfer functions, considering the influence of the external temperature, the action of parasitic capacity on the behaviour of the transistor at high frequencies, or will it be enough to provide a rougher description in terms of Kirchoff's laws?

The first answer is that it is important to remember that you are not building a simulator, but a system for troubleshooting, that should be able to grasp only the minimal set of aspects relevant to the diagnostic requirement.

The second answer is that you must assume that your physical system will work properly, i.e. that there are no malfunctions due to the system design itself, because the validation of the system design lies beyond the scope of your investigation.

The third answer is that it all depends on the kind of diagnostics you are considering, whether it is the one at system level, where the goal is to identify a major item that has failed, or the one at component level where the aim is to identify the component responsible for the malfunction.

The approach that we are describing in this paper has been conceived for the "highest" possible level, sometimes called the "first" level, where the aim is to replace the failed item to recover the system readiness in the shortest possible time. Our approach has been tested for troubleshooting at system level, where it has proven to be effective in fault isolation, while it has not been tested for other kinds of diagnostic.

ADAM support system

In the course of the present research project, the first phase was devoted to building a version of a support system based on the heuristic approach. The software system was based on diagnostic trees, i.e. it was basically a tool for modeling diagnostic knowledge by using the formalism of diagnostic trees. For more information see [1].

The second phase of the project has led to a second support system capable of inferring the troubleshooting procedure starting from a deep model of a physical system, as described in the previous chapter. At present, system validation is still ongoing.

In this chapter we will give an overview of the ADAM system interface and of its main functionalities.

In figure 3 the interface is represented, and as you can see it is provided with a set of graphical tools (toolbar) that allow the user to represent on the screen all the entities in the deep model. So, from the user point of view, the deep model will look like a schematic diagram, where all the entities will be represented by means of boxes and connected by means of lines representing wirings, pipes, databuses and so on.

Moreover, each object in the deep model will be associated to a set of clauses, as described in the previous chapters, and will be provided with information like defect rate, component status, replacement time and so on.

After the deep model generation, the user will be able to run a troubleshooting session. At first he will list all the symptoms of the malfunctions detected either by the groundcrew or by the pilot. This will be possible by mouse and menus.

The next step will be to execute the troubleshooting algorithm that will generate the possible solutions for the problem. The output for this process is a list of pairs component/failure states that will identify a possible system fault. In our example it could be "Circuit Breaker - Open".

ADAM is provided with a graphical explanation mechanism: each time that a failure is proposed for the symptoms detected, the graphical representation of the failed item will change in color (e.g. it will turn to red) and the user will be able to identify on the screen the path from the initial symptoms to the failed component.

An example

Let us suppose that the failure message on the groundcrew display is: "Detector failed".

This kind of message is generated when the control unit detects a loss of current on the line that goes from the electrical power to the control unit, via the integrity switch. This could be due to a failure of the fire detector, but also to a failure on the line that brings electrical power from the generator to the control unit.

At this point a maintenance operator, possibly low skilled on the system under consideration, will have to understand how such a warning message could have been generated. Generally this is not a trivial operation, and could mean browsing through manuals, consulting diagrams and so on.

The symptom can be due to many causes: the electrical power can be off, the circuit breaker can be open, the integrity switch can be open, the wiring can be open-circuit and the problem could even be in the control unit. The operator must choose an action among all the possible ones, and the problem is what to check first.

What the user needs at this stage is a list of possible actions to perform on the fire protection system, ranked according to a minimum cost criterion.
Such a criterion must consider the defect rate and the cost of the action itself. Such two criteria may be contradictory, i.e. the cheapest recovery action (in terms of time and of spare parts cost) may be the least likely to occur and vice versa. At the time being we simply have chosen to rank the recovery actions by means of the quotient between the repair time and the defect rate, the smaller is the number the best is the associated repair action. So, after the symptom description and the system run, the final output is a list of possible failures ordered from best to worst.

The advantage of a support system in this example is clear: the failure report "Detector Failed" could be either unclear or even be misleading, because it can originate from four different causes. ADAM is useful for:

1. finding the proper set of possible failed items starting from the groundcrew indication,

2. ranking the list of actions on the failed items in a best probability order for maintenance operations.

The user interface

In this chapter we will give a short description of the user interface that is represented in figure 3.

Basically the tool has graphical capability for representing the deep model by means of something very close to the usual schematic diagrams. So the white area in the center of the screen (canvas) is used for drawing objects, usually square boxes, and connecting them by means of lines which represent wires, pipes, databuses etc.

The toolbar is used for selecting the entity to be drawn, or the kind of link, while the pull-down menus is used for activating the system functionalities, i.e. for saving/loading deep models, for selecting symptoms and so on.

The canvas is also used for the explanation.

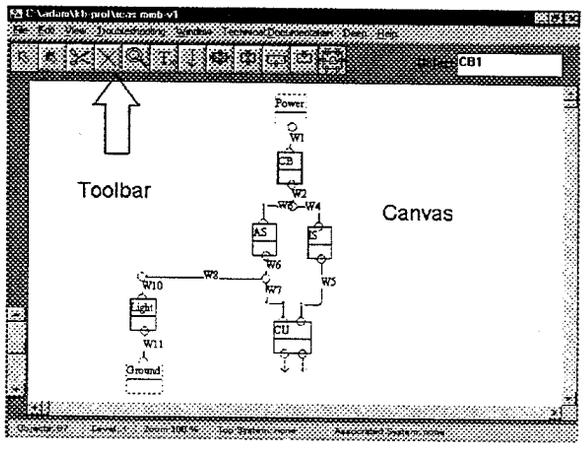

FIGURE 3 - ADAM Interface

The troubleshooting algorithm

We will try in this chapter to give an idea on how our model-based algorithm works, without going into deep detail.

The ADAM algorithm accepts two kinds of symptoms, i.e. those that must be "explained" by the system and those that are only additional data for reducing the search space, and that must not be contradicted.

Our algorithm is able to generate sets of hypotheses and to delete those sets that are in contrast. Hypothesis are generated in a backward strategy, starting from symptoms and "going through" the system components. Each time that a new component is reached, new hypotheses are generated about the component, and the algorithm will reject those that are inconsistent.

The result of the computation is a lattice structure that keeps track of the assumptions made about each component.

At the end it will be possible to associate to each symptom many sets of assumptions. Each set represents an explanation, i.e. a diagnosis for the symptom selected.

Implementation notes and future developments

The present ADAM software release has been totally developed in C++ and runs on an ordinary PC platform.

Future efforts on this research project will be devoted to building component libraries so as to allow the user to build system deep-models without having to write most parts of their clauses, and to improve the formal language for clause description.

Final remarks

A support system for unscheduled maintenance should be able to provide the system designer with precise knowledge of the troubleshooting procedure during the design phase itself, and to collect the experience gained by maintainers.

The two aspects mentioned here are covered by the two different ADAM releases: the heuristic version, i.e. the one based on diagnostic trees, which is described in [1], is suitable for storing troubleshooting procedures and for modifying them easily and quickly, while the model-based version, which has been extensively described in this paper, is suitable for automatically generating diagnostic trees. By combining the two systems is possible to generate an initial version of diagnostic procedures and then to modify them taking practical experience into account.

GIANARELLI

Acknowledgments

This research project would not have been possible without the support of many colleagues in the Alenia Aerospazio - Aeronautics Division.

We thank Stefania Converso, Emanuela Damiani, Oriano Fadini, Ashley Hogg and Raffaele Prete for their insightful discussions and related work that bear on the material presented here.

References

1. S.Damiani, F.Ricci, E.Valfré - "ADAM Aircraft Diagnostic and Maintenance" in Proceedings of the third Conference of the Italian Association for Artificial Intelligence - Turin Italy - October 1993

2. L.Console, P.Torasso - "A Spectrum of Logical Definitions of Model-Based Diagnosis" in Readings in Model-Based Diagnosis - Morgan-Kaufmann, San Francisco, California, 1992

3. L.Console, G.Friedrich - "Model-Based Diagnosis" in Annals of Mathematics and Artificial Intelligence - J.C.Baltzer AG - Swiss - 1994

GIRARDELLI