Methodology and Application of Meta-Diagnosis on Avionics Test Benches

R. Cossé[1,2], D. Berdjag[2], S. Piechowiak[2], D. Duvivier[2], C. Gaurel[1]
[1]AIRBUS HELICOPTERS, Marseille International Airport, 13725 Marignane France
{ronan.cosse, christian.gaurel}@airbus.com
[2]LAMIH UMR CNRS 8201, University of Valenciennes, 59313 Valenciennes France
{denis.berdjag, sylvain.piechowiak, david.duvivier}@univ-valenciennes.fr

Abstract

This paper addresses Model Based Diagnosis for the test of avionics systems that combines aeronautic computers with simulation software. Just like the aircraft, those systems are complex since additional tools, equipments and simulation software are needed to be consistent with the test requirements. We propose a structural diagnostic framework based on the lattice concept to reduce the time of unscheduled maintenance when the tests cannot be performed. Here, we also describe a diagnosis algorithm that is based on the formal lattice description and designed for test systems. The benefits is to capture the system structure and communication specificities to diagnose the configuration, the equipments, the connections, and the simulation software.

1 Introduction

Avionics systems are complex since tens of subsystems and components interact to achieve required functions. Existing devices for aircraft fault monitoring are based on dedicated avionics functions but the existing solutions are insufficiently flexible for test systems and can be improved. In [1], the framework of an health management algorithms for maintenance is described and implemented on an aircraft. In [2], the diagnostic of avionics equipments is performed through dynamic fault trees. To prevent important failures on the aircraft, avionics systems are checked on rigs called Avionics Test Bench (ATB) composed of the avionics equipments and flight simulation software.
The environment of the ATB needs to be compliant with the configuration of the avionics equipments. Faults of the ATB can concern the avionics equipments, their configurations, or the ATB itself i.e the movable connections and the simulation software. Since it does not exist monitoring functions of the ATB itself, a new method needs to be applied to prevent long periods of unavailability. In fact, during the development of embedded softwares, its architecture and the test environment surrounding the ATB are redesigned by adapting the test means to the specification's requirements. Since the ATB is a test system, and the main knowledge are based on its embedded systems, we need a new approach to deal with the ATB issues. As the embedded systems are already tested on the ATB, and the test results are used to focus on the ATB issues thanks to a new representation based on the model of the test system, the diagnosis of the ATB is what we call a meta-diagnosis.
Many diagnosis approaches have been proposed to deal with specific avionics problems. Two different classes of representation are applied: data-based diagnosis or model-based diagnosis. The first one, as studied by Berdjag et al. [3] is used to recognize faulty behaviors of an Inertial Reference System (IRS) thanks to normal or faulty categories of input/output data. In this work, data fusion of outputs sensors is computed to eliminate faulty sources. In [2], the time dependency is introduced in data of failure messages to improve problems detection.
In Model Based Diagnosis (MBD), Kuntz et al. [4] have studied an avionics system using minimal cuts notions. Belard et al. have defined a new approach based on the MBD hypotheses called Meta-Diagnosis in [5] dealing with models issues. Berdjag et al. [6] present an algebraic decomposition of the model to reduce the complexity of the required model-based diagnosers. Giap [7] has proposed a formalism of an iterative process to give a solution when models are not complete but it lacks of applications on more complex industrial systems. Nevertheless, it gives clues for an iterative diagnosis. Another diagnostic software has been developed by Pulido et al. in [8] to perform consistency-based diagnosis of dynamic system simulating diagnosis scenarios. The architecture is quite novel and is applied to the three-tank system.
Structural approaches as graph theory are also popular for MBD to describe the structure of the system as with Bayesian Networks in [9]. They enable us to incorporate the system complexity as with the lattice concept to integrate the sub-models dependencies. For example, in [10], the lattice model represents fault modes to compute testable subsystems from redundancy equations. We want to get the main ideas that will serve our proposal. To our knowledge, there is no method for the diagnostic of test systems based on embedded softwares behaviour. Moreover, our proposition has been adapted from embedded systems to the ATB behaviour. Its complexity is relevant to the objectives of the avionics embedded systems certification, as for example high levels of safety requirements, or the simulation of specific test conditions. In our model, we must consider the fact that our representation must put forward the ATB behaviour in case of failures concerning embedded systems, connections, communications, simulation softwares and all settings to configure the test. Considering those features, the high number of needed ATB reconfigurations, it is proposed a structural representation associated with hierarchical verifications that reduce the faulty candidates. The motiva-

tion of the proposed meta-diagnosis approach was presented in [11]. Here, we propose an extended diagnosis methodology originally defined by De Kleer, Williams [12], [13] and Davis [14] and we present a software implementation running on a real ATB. It differs from the Belard et al.'s meta-diagnosis definition because the ATB is still defined as the main system under study. Here, we extend the diagnostic-world tools for a specific system and due to the lack of knowledge and data in case of issues, our proposal is based on a MBD representation with a structural and functional decomposition without fault models.
First, we describe the diagnostic framework, the lattice-based representation used to model the ATB system and the diagnostic algorithm. In the third section, we provide a description of the ATB and the application of the lattice concept. In the fourth section, we illustrate the approach with a case study of the ATB. In the final section, we describe the development of a software application to perform automatically the ATB diagnosis.

2 Diagnostic framework

2.1 System representation

The system is composed of several subsystems that interact together to achieve a global function. The decompositions into subsystems is guided by the communication between components to fulfill this goal. Partitions are used to decompose the system into functional and communications categories. So, there are two classes of partitions: the partitions that represent the structure and the connections of the system; and the partitions that represent the functions of the system. As an example, P_1 is associated with a functionality of the system $P_1 = \{\sigma_1; \sigma_2\}$, $\sigma_1 = \{C_1\}$ and $\sigma_2 = \{C_2, C_3\}$. If a problem appears, i.e the functionality is not performed, then a fault is detected for this partition P and symptoms are seen and linked to subsystems σ.
In the following paragraphs, we use the following notation: P for a partition, σ for a subsystem and c_i for a component. $S = \{c_i, i \in [1, n]\}$ is the set of all the n components of a system. We note Σ the set of all subsystems, i.e the power set of components. A partition P is a set of n_p subsystems $\sigma_i \in \Sigma$: $P = \{\sigma_i, i \in [1, n_p] | \forall i \neq j; \sigma_i \cap \sigma_j = \emptyset$, and $\bigcup_{i=1}^{n_p} \sigma_i = S\}$. We note $\mathscr{P}$ the set of all partitions.
We recall the definition 1 of inclusion relation between partitions and the definition 2 of multiplication.

Definition 1. *Two partitions P_1 and P_1 are said to be in inclusion relation $P_1 \subseteq P_2$ if and only if every subsystems of P_1 is contained in a subsystem of P_2. The relation $\subseteq$ means that P_1 is a sub-partition of P_2.*

Definition 2. *The subsystems σ_k of the multiplication of two partitions $P = \{\sigma_i, i \in [1, n_p]\}$ and $Q = \{\sigma_j, i \in [1, n_q]\}$ are defined by: $\forall \sigma_k \in P \times Q, \exists \sigma_i \in P, \exists \sigma_j \in Q, \sigma_k = \sigma_i \cap \sigma_j$.*

This operation is used to order subsystems with respect to the proposed diagnostic algorithm. The inclusion relation $\subseteq$ is used to organize the components with the lattice concept $\mathscr{L}(\Sigma, \subseteq)$ with a partial ordering relation. It is different from the concept of partially ordered set (poset) because the arrangement of elements is not based on sets but on partitions.

2.2 Diagnostic function

A basic diagnostic function is defined to help the diagnosis: the $check$ function. Depending on the granularity, the $check$ function is applied on a component, a subsystem or a partition. First, the $checkC$ function is used to determine if a component is faulty or not. However, we do not know precisely how a unique component behaves regarding a fault. So we need to define the $checkS$ function of a subsystem. The behaviour of a faulty subsystem may also not be sufficient to explain a fault. In fact, subsystems are interconnected making the system structure and the partitioning concept allows us to focus on different levels of abstraction that we call granularities. In our study, we only focus on faults with observable and measurable symptoms. These faults can only be localized by testing a functionality on a specific architecture. That is why, functional and structural partitions are used to decompose the system into testable partitions.

Definition 3. *The checkC function of a component c_i is defined by:*
$checkC : COMPS \rightarrow \{0, 1, -1\}$ s.a $checkC(c) = 0$ if the component c is faulty, $checkC(c) = 1$ if the component c is unfaulty and $checkC(c) = -1$ if the component state is unknown.

Definition 4. *The checkP function of a partition P is defined by:*
$checkP : \mathscr{P} \rightarrow \{0, 1, -1\}$ s.a $checkP(P) = 1 \Leftrightarrow \forall \sigma_i \in P, checkS(\sigma_i) = 1$, $checkP(P) = 0 \Leftrightarrow \exists \sigma_i \in P, checkS(\sigma_i) = 0$, and $checkP(P) = -1 \Leftrightarrow$ the checked value is unknown.
Some partitions cannot be checked. The set of possible checked partitions is $Cons$. It defined a constraint. A constraint $Cons$ is a subset of $\mathscr{P}$ s.a: $\forall P \in Cons, checkP(P) \neq -1$.

Once the $checkP$ value of a partition is known, we have to define the $checkS$ function of subsystems that are not singletons $\sigma_i \neq \{c_i\}$. If the partition is faulty, either it exists a component $c_i \in \sigma_i$ such as $checkC(c_i) = 0$, or the communication between the components in σ_i is faulty. This is modeled by $checkCom(\sigma_i) = 0$. If the partition is unfaulty, then all communications between the components in $\sigma_i \neq \{c_i\}$ are unfaulty and all singletons $\sigma_i = \{c_i\}$ are unfaulty.

Definition 5. *The $checkCom$ function of a subsystem $\sigma_i \subseteq COMPS$ is defined by:*
$checkCom : \Sigma \rightarrow \{0, 1, -1\}$ s.a $checkCom(\sigma_i) = 1 \Leftrightarrow$
the communication between components in σ_i is unfaulty;
$checkCom(\sigma_i) = 0 \Leftrightarrow$
the communications between components in σ_i is faulty.

To help the diagnosis of the system, we decompose it into subsystems and we introduce the $checkS$ function of a subsystem $\sigma_i \subseteq COMPS$ defined by:

Definition 6. *$checkS : \Sigma \rightarrow \{0, 1, -1\}$ s.a $checkS(\sigma_i) = 1 \Leftrightarrow \forall c_i \in \sigma_i, checkC(c_i) = 1 \wedge checkCom(\sigma_i) = 1$; $checkS(\sigma_i) = 0 \Leftrightarrow \exists c_i \in \sigma_i, checkC(c_i) = 0 \vee checkCom(\sigma_i) = 0$ and $checkS(\sigma_i) = -1 \Leftrightarrow \exists c_i \in \sigma_i, checkC(c_i) = -1 \wedge checkCom(\sigma_i) = -1$.*

With the above definitions, it is now time to define the diagnosis problem. Given a system representation with the lattice concept $\mathscr{L}(\Sigma, \subseteq)$ and the set of constraints $Cons =$

$\{P \in \mathscr{P}, checkP(P) \neq -1\}$, the problem is defined by the consistency between $\mathscr{L}(\Sigma, \subseteq)$ that contains the system representation, and $Cons$ that describes system issues.

Definition 7. *The problem formulation is to find the faulty components whose current state may explain the constraints. It is defined as a function $DIAG(\mathscr{L}(\Sigma, \subseteq))$ under the constraints $Cons$.*

There are two kinds of faults: the fault of a component C_i modeled with $checkC(C_i) = 0$, and the communication fault of a subsystem $\sigma_i = \{C_i, C_j, ...\}$ modeled with $checkCom(\sigma_i) = 0$. With the P_1 partition, suppose that C_2 and C_3 are linked with an ARINC 429 link that is not working. The constraint is $checkP(P_1) = 0$ because the global function is broken. The reason is that $checkCom(\sigma_2) = 0$. Knowing that $checkCom(\sigma_2) = 0$ for the P_1 functionality is giving the information to fix the system.

2.3 Diagnostic algorithm

It is now necessary to introduce a diagnostic method whose aim is to solve the above problem. The algorithm is based on the following proposition that extends the verification from the multiplication of partitions to partitions, see Proposition 1. Then, a functional verification is propagated from partitions to subsystems, and from subsystems to components.

Proposition 1. $\forall P, Q \in \mathscr{P}^2, checkP(P \times Q) = 0 \Rightarrow checkP(P) = 0 \wedge checkP(Q) = 0$.

In order to increase the readability of the algorithm, it has been split into three: $DIAG(\mathscr{L}(\Sigma, \subseteq))$ is the main algorithm, it initializes the framework with the partitions of the system $\{p_i, i \in [1, n]\}$ and the constraints $Cons = \{P \in \mathscr{P}, checkP(P) \neq x\}$.
$FindFaultyElements$ checks the partitions that are defined as a constraint. If the checked value of a partition p_{mult} is faulty (*resp.* unfaulty), we add it to the faulty (*resp.* unfaulty) partitions set P^- (*resp.* P^+), and every subsystem σ_i of the partition is possibly faulty (*resp.* unfaulty), we add it in Σ^+, (*resp.* Σ^-). If another partition p_{mult} can help to get more faulty or unfaulty components, a new constraint is proposed and added to $NCons$.
$Verification$ is used to check the possible components that may be faulty, i.e include in F_c with the $checkC$ function, and the communication of the subsystems in Σ^- with the $checkCom$ function.

Two functions have been introduced: the $checkP(p_i)$ value of a partition p_i and the $CheckCom(\sigma_i)$ of a subsystem. Their values can be automatically computed thanks to a program developed on the system to automate the diagnosis. This is performed by the GET function whose purpose is to model the computation of $checkP(p_i)$ or $CheckCom(\sigma_i)$.

2.4 Formal example

In order to illustrate the problem formulation and the diagnostic algorithm, a formal example is provided. It is composed of eight components $\{C_i,\ i \in [1, 8]\}$ organized into three partitions:
P_1 = { {C_1,C_2, C_3,C_4}, {C_5,C_6, C_7,C_8}},
P_2 = { {C_1,C_2}, {C_3,C_4,C_5,C_6,C_7,C_8}},
P_3 ={{C_1}, {C_2,C_4,C_6,C_8}, {C_3,C_5,C_7}}.
P_3 describes the topology of the system. P_1 and P_2 describe functionalities. We set the C_2 component as faulty. The idea is to combine the topology of the system with its functionalities to find the faulty component or subsystem. A choice

Algorithm 1: $DIAG(\mathscr{L}(\Sigma, \subseteq))$

```
Input: $d = \{p_i, i \in [1, n]\}, Cons = \{cons_i\}$
Output: $\Delta(Diagnosis)$
Global variables: $End$
$F_c(faulty\ components), U_c(unfaulty\ components),$
$\Sigma^-(faulty\ subsystems), \Sigma^+(unfaulty\ subsystems),$
$P^-(faulty\ partitions), P^+(unfaulty\ partitions)$
$\Delta, F_c, U_c, P^+, P^-, \Sigma^-, \Sigma^+ \leftarrow \{\}$; $End \leftarrow false$;
$NCons \leftarrow \{\}$;
while $\neg End$ do
    $FindFaultySubsystems(d, Cons)$;
    $Verification(F_c, \Sigma^-)$;
    if $\neg End$ then
        foreach $p_i \in NCons$ do
            GET $checkP(p_i)$
            $Cons \leftarrow Cons \cup \{p_i\}$
```

Algorithm 2: $FindFaultyElements$

```
Input: $d = \{p_i\}, Cons = \{cons_i\}$
Outputs: $F_c, P^-, \Sigma^-, \Sigma^+$
foreach $(p_j, p_k) \in P^2 : p_i \neq p_j$ do
    $p_{mult} \leftarrow p_j \times p_k$
    if $p_{mult} \in Cons$ then
        if $checkP(p_{mult}) = 0$ then
            $P^- \leftarrow P^- \cup \{p_i\}$
            foreach $\sigma_i \in p_i$ do
                foreach $c_k \in U_c$ do
                    $\sigma_i \leftarrow \sigma_i \setminus \{c_k\}$
                if $\sigma_i = \{c_i\}$ then
                    $F_c \leftarrow F_c \cup \sigma_i$
                else if $\sigma_i \notin \Sigma^+$ then
                    $\Sigma^- \leftarrow \Sigma^- \cup \{\sigma_i\}$
        if $checkP(p_{mult}) = 1$ then
            $P^+ \leftarrow P^+ \cup \{p_i\}$
            foreach $\sigma_i \in p_i$ do
                if $\sigma_i = \{c_i\}$ then
                    $U_c \leftarrow U_c \cup \sigma_i$
                else
                    $\Sigma^+ \leftarrow \Sigma^+ \cup \{\sigma_i\}$
    if $p_{mult} \notin Cons$ then
        if $\exists\{c_i\} \in p_{mult}$ then
            if $\neg(c_i \in U_c \cup F_c)$ then
                $NCons \leftarrow NCons \cup \{p_{mult}\}$
```

function is introduced to choose the next topology and the next functionality to be tested. It is guided by the minimum of tests to perform in order to fix the system. For a set of partitions $\mathscr{P}$, we define $Choose : \{\mathscr{P}\} \rightarrow \mathscr{P} \times \mathscr{P}$.
As the two functionalities are modeled by P_1 and P_2, and the the topology is modeled by P_3, we have two possibilities. We assume that P_2 is prior to P_1, the first iteration is defined with $Choose(\mathscr{P})$=(P_1, P_3). We begin with $checkP(P_1 {\times} P_3)$ = 0, s.a $P_1 \times P_3$ = { { C_1 }, {C_2,C_4}, {C_3}, {C_6,C_8}, {C_5,C_7}}. The possible faulty component are C_1 and C_3. We check the C_1 and C_3 components and

Algorithm 3: $Verification$

```
Inputs: $F_c$
Outputs: $\Delta$ $F_c$, $U_c$, $End$
Initialization: $\sigma_+, \sigma_- \leftarrow I$;
foreach $c_i \in F_c$ do
    if $checkC(c_i) = 0$ then
        $\Delta \leftarrow \Delta \cup \{c_i\}$
        $End \leftarrow true$
    else
        $F_c \leftarrow F_c \setminus \{c_i\}$
        $U_c \leftarrow U_c \cup \{c_i\}$
foreach $\Sigma_i \in \Sigma^-$ do
    GET $checkCom(\Sigma_i)$
    if $checkCom(\Sigma_i) = 0$ then
        $\Delta \leftarrow \Delta \cup \{\Sigma_i\}$
        $End \leftarrow true$
    else
        $\Sigma^- \leftarrow \Sigma^- \setminus \{\Sigma_i\}$
        $\Sigma^+ \leftarrow \Sigma^+ \cup \{\Sigma_i\}$
```

find them as unfaulty, see Tables 1. The possible faulty subsystems are $\{C_2, C_4\}$, $\{C_6, C_8\}$ and $\{C_5, C_7\}$ and they are unfaulty. The diagnosis is not sufficient, we must relax the constraint $P_2 \times P_3$.
The second iteration is defined with $Choose(\mathscr{P})$=(P_2, P_3), s.a $P_2 \times P_3$ = $\{\{C_1\}, \{C_2\}, \{C_4,C_6,C_8\}, \{C_3,C_5,C_7\}\}$. We get $checkP(P_2 \times P_3) = 0$, the possible faulty components are C_1 and C_2 but C_1 has already been checked in the previous iteration. So, the possible faulty subsystems are $\{C_3,C_5,C_7\}$ and $\{C_4,C_6,C_8\}$. We check the C_2 component and find it as faulty. For this example, the computed faulty or unfaulty components is, see Table 2, C_2 in $P_2 \times P_3$.
If no components has been found faulty, the upper topological level is treated i.e subsystems: $\{C_2,C_4\}$, $\{C_6,C_8\}$, $\{C_5,C_7\}$, $\{C_4,C_6,C_8\}$ and $\{C_3,C_5,C_7\}\}$. Here, they are unfaulty.

Components	CheckC
C_1	1
C_2	-1
C_3	1
C_4	-1
C_5	-1
C_6	-1
C_7	-1
C_8	-1

Table 1: Diagnostic results for components in $P_1 \times P_3$

The method has permitted to detect quickly the faulty component using functional partition and a structural partitioning. Thanks to this result, possible faults regarding either the topology or the functionality are checked.

3 The Automatic Test Benchmark

3.1 Avionics system

The avionics system of the NH90 helicopter is designed to support multiple hardware and software platforms from more than twelve national customers in over twenty different basic helicopter configurations. The NH90 Avionics System consists of two major subsystems: the CORE System and the MISSION System. A computer is the bus controller and manages each subsystem communications: the Core Management Computer (CMC) for the CORE System and the Mission Tactical Computer (MTC) for the MISSION System. Each computer is connected to one or both subsystems via a multiplex data bus (MIL-STD-1553), point to point connections (ARINC429) and serial RS-485 lines. Additional redundant computers are used as backup. One of the two CMC is the Bus Controller (BC) of the CORE multiplex data bus. The avionics system of the ATB is composed of fourteen computers and the above connections: two CMC: $c_1 = CMC1$ and $c_2 = CMC2$; two Plant Management Computer (PMC): $c_3 = PMC1$ and $c_4 = PMC2$; five Multifunction Display (MFD): $c_5 = MFD1$, $c_6 = MFD2$, $c_7 = MFD3$, $c_8 = MFD4$, $c_9 = MFD5$; two Display and Keyboard Unit (DKU): $c_{10} = DKU1$, $c_{11} = DKU2$; two IRS: $c_{12} = IRS1$, $c_{13} = IRS2$; one Radio Altimeter (RA): $c_{14} = RA$. Formally, $COMPS_{ATB} = \{c_i, i \in [1, 14]\}$.
The avionics system under test $COMPS_{SUT}$ is a subsystem of $COMPS_{ATB}$. It is described Figure 1. $COMPS_{SUT} = \{c_1, c_2, c_3, c_4, c_5, c_{10}, c_{12}, c_{14}\}$. For the rest of the article, $COMPS_{SUT}$ will be the primary system under study.

Components	CheckC
C_1	1
C_2	0
C_3	1
C_4	-1
C_5	-1
C_6	-1
C_7	-1
C_8	-1

Table 2: Diagnostic results for components in $P_2 \times P_3$

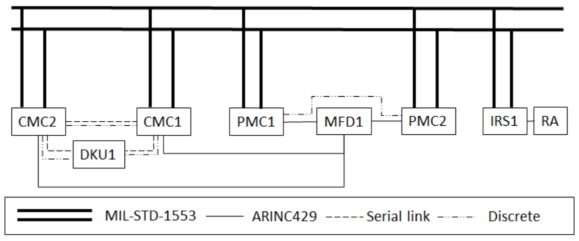


Figure 1: Architecture of the avionics subsystem

From	To	Messages	Subsystems
$DKU1$	$CMC1$	Mode on	$\sigma_{Serial1}$
$CMC1$	$IRS1$	Mode on	σ_{MIL}
$IRS1$	RA	Mode on	$\sigma_{NAV}; \sigma_{ARINC}$
RA	$IRS1$	Alert	$\sigma_{NAV}; \sigma_{ARINC}$
$IRS1$	$CMC1$	Alert	$\sigma_{MIL}; \sigma_{NAV}$
$CMC1$	$DKU1$	Alert	$\sigma_{Serial1}; \sigma_{NAV}$

Table 3: Messages

The PMC is used to monitor the status of all the avionics computers. It displays the alert informations on the MFD. We define the performances partition p_{PERF} = $\{\sigma_{PERF},\sigma_{\neg PERF}\}$ with:
σ_{PERF} = $\{PMC1, PMC2, RA, IRS1, MFD1\}$
$\sigma_{\neg PERF}$ = $\{CMC1, CMC2, DKU1\}$ and the navigation partition p_{NAV}= $\{\sigma_{NAV},\sigma_{\neg NAV}\}$ with:
σ_{NAV} = $\{ RA, IRS1, MFD1\}$
$\sigma_{\neg NAV}$ = $\{CMC1, CMC2, DKU1, PMC1, PMC2\}$.
The test consists in the simulation of a high roll. Normally the RA should be deactivated above the value of forty degrees. The procedure contains the following actions: engage the RA with the $DKU1$; simulating a roll of 50 degrees; check that the RA functionality is deactivated on the $DKU1$. Several messages are sent to achieve this functionality, see Table 3, defining a data-flow for two messages : "Mode on" and "Alert" messages: from $DKU1$ to $CMC1$ via serial communication to activate the radioaltimeter's specific mode ("Mode on" message); from $CMC1$ to $IRS1$ via MIL-STD-1553 communication to relay the activation information; from $IRS1$ to RA via ARINC communication to send a request to the RA to get the roll angle; from RA to $IRS1$ via ARINC communication to send the response to the IRS that compute the angle; from $IRS1$ to $CMC1$ via ARINC communication, from CMC to DKU via serial communication to display the alert and disable the functionality ("Alert" message).

3.2 System Under Test (SUT) decomposition

The ATB is used to perform the realization of the avionics functions with the necessary equipments and a simulated environment needed to check the system specification.

The ATB is described as a structural decomposition with components subsets. These sets provide partitions of the whole system. We define subsystems σ_i and the partitions p_i with regards to the connections of the avionics system of Figure 1, the serial communication:

$$\begin{aligned}\sigma_{Serial1} &= \{CMC1, CMC2, DKU1\}\\ \sigma_{Serial2} &= \{PMC1, PMC2\}\\ \sigma_{\neg Serial} &= \{MFD1, IRS1, RA\}\\ p_{Serial} &= \{\sigma_{Serial1}; \sigma_{Serial2}; \sigma_{\neg Serial}\}\end{aligned}$$

the ARINC communications:

$$\begin{aligned}\sigma_{ARINC} &= \{CMC1, CMC2, PMC1, PMC2,\\ &\quad MFD1, IRS1, RA\}\\ \sigma_{\neg ARINC} &= \{DKU1\}\\ p_{ARINC} &= \{\sigma_{ARINC}; \sigma_{\neg ARINC}\}\end{aligned}$$

the MIL-STD-1553 communications:

$$\begin{aligned}\sigma_{MIL} &= \{CMC1, CMC2, PMC1, PMC2, IRS1\}\\ \sigma_{\neg MIL} &= \{MFD1, DKU1, RA\}\\ p_{MIL} &= \{\sigma_{MIL}; \sigma_{\neg MIL}\}\end{aligned}$$

The above partitions describe the topology of the problem. We classify the partitions into two categories: functional partitions and communication partitions. The functional partitions contain the subsystems that compute and send the informations. The communication partitions contain the subsystems that relay these informations. In our example, the navigation functionality is tested. Functional partition are: $\{p_{NAV}, p_{PERF}\}$, connection partitions are: $\{p_{MIL}, p_{Serial}, p_{ARINC}\}$. We need to define additional partitions that can be checked with the *check* function on the system thanks to this representation:
$p_{NAV.MIL}$ = $p_{NAV} \times p_{MIL}$ = $\{\{MFD1, RA\};\{IRS1\}; \{CMC1, CMC2, PMC1, PMC2\};\{DKU1\}\}$;
$p_{NAV.Serial}$ = $p_{NAV} \times p_{Serial}$ = $\{\{CMC1, CMC2, DKU1\}; \{PMC1, PMC2\}; \{MFD1, IRS1, RA\}\}$;
$p_{NAV.ARINC}$ = $p_{NAV} \times p_{ARINC}$ = $\{\{MFD1, IRS1, RA\}; \{CMC1, CMC2, PMC1, PMC2\}; \{DKU1\}\}$.

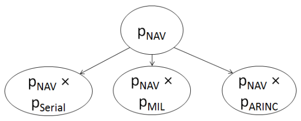


Figure 2: Navigation function decomposition with $d_{protocol}$

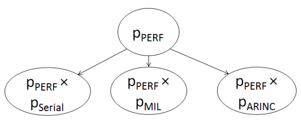


Figure 3: Performance function decomposition with $d_{protocol}$

The performance function can give insights about the fault. We compute the partitions with this functionality:
$p_{PERF.MIL}$ = $p_{PERF} \times p_{MIL}$ = $\{ \{MFD1, RA\}; \{DKU1\}; \{CMC1, CMC2\}; \{PMC1, PMC2, IRS1\} \}$
$p_{PERF.Serial}$=$p_{PERF} \times p_{Serial}$ = $\{ \{CMC1, CMC2, DKU1\}; \{PMC1, PMC2\}; \{MFD1, IRS1, RA\} \}$
$p_{PERF.ARINC}$ = $p_{PERF} \times p_{ARINC}$ = $\{ \{ PMC1, PMC2, MFD1, IRS1, RA\};\{CMC1, CMC2\}; \{DKU1\} \}$.
Those partitions will serve to improve the diagnosis.

3.3 Outlooks about the decompositions

We describe an iterative method to update the diagnostic result by providing new topologies of the system. We need to get precise observations to find the faulty components. The subsystems are computed with the framework of the previous section.
Given the components, the messages sent between them, and the protocol of these messages, we can obtain an overview of the system decomposition: p_{SUT} can be decomposed into $d_{protocol} = \{p_{SUT} \times p_{MIL}; p_{SUT} \times p_{Serial}; p_{SUT} \times p_{ARINC}\}$. This hierarchical structure is provided with a dependency graph, see Figures 2 and 3.

The following partitions are used:
$\sigma_{com_1} = \{\{DKU1, CMC1, IRS1, RA\}\}$;
$\sigma_{\neg com_1} = \{\{MFD1, CMC2, PMC1, PMC2\}\}$;
$p_{com_1} = \{\sigma_{com_1}, \sigma_{\neg com_1}\}$.

The path of the informations "RA mode on" and "RA alert" on copilot side defines another decomposition: σ_{com_2} = $\{\{CMC2, IRS1, RA, DKU1\}\}$; $\sigma_{\neg com_2}$ = $\{\{MFD1, CMC1, PMC1, PMC2\}\}$; p_{com_2} = $\{\sigma_{com_2}, \sigma_{\neg com_2}\}$.

We describe the decomposition $d_{com} = \{p_{com1}, p_{com2}\}$ on Figures 4 and 5. We compute partitions with the navigability functionality and this structural decomposition:
$p_{NAV.com1}$ = $p_{NAV} \times p_{com1}$ = $\{\{RA, IRS1\}; \{MFD1\}; \{CMC1, DKU1\}; \{CMC2, PMC1, PMC2\}\}$;
$p_{NAV.com2}$ = $p_{NAV} \times p_{com2}$ = $\{\{RA, IRS1\}; \{DKU1, CMC2\}; \{MFD1\}; \{CMC1, PMC1, PMC2\}\}$;
$p_{PERF.com1}$ = $p_{PERF} \times p_{com1}$ = $\{\{RA, IRS1\}; \{CMC2\}; \{CMC1, DKU1\}; \{MFD1, PMC1, PMC2\}\}$;
$p_{PERF.com2}$ = $p_{PERF} \times p_{com2}$ = $\{\{RA, IRS1\}; \{DKU1, CMC2\}; \{CMC1\}; \{MFD1, PMC1, PMC2\}\}$.

4 Illustration of the Meta-Diagnostic Approach

4.1 Application of the meta-diagnosis approach

An iterative approach is very helpful in this case of distributed systems since diagnosis can use new subsystems and partitions. The results of the diagnosis are re-injected in the upper system to refine the results.

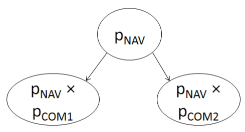


Figure 4: Navigation function decomposition with d_{com}

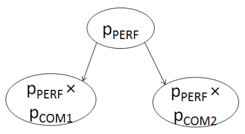


Figure 5: Performance function decomposition with d_{com}

The first symptom is the misbehavior of the navigation functionality. We describe the iterations of the algorithms with two topologies. We have launched the meta-diagnostic algorithm with the topology: $d_{NAV.protocol}$ = $\{p_{NAV.MIL}$,$p_{NAV.ARINC}$,$p_{NAV.SERIAL}\}$ and $d_{NAV.com}$ = $\{p_{NAV.com1}$, $p_{NAV.com2}\}$. The constraint is $CONS = \{checkP(p_i), \forall p_i \in d_{NAV.protocol} \cup d_{NAV.com}\}$. The iterations of the algorithms are described in Tables 4, and 5.

p_i	$checkP(p_i)$	U_c	F_c
$p_{NAV.ARINC}$	0	$\emptyset$	$\{DKU1\}$
$p_{NAV.SERIAL}$	1	$\emptyset$	$\{DKU1\}$
$p_{NAV.MIL}$	0	$\emptyset$	$\{IRS1, DKU1\}$

Table 4: Iterations of $CheckMultiplicationPartition$ with $d_{protocol}$

The third step gives a state of the components in F_c set that can be faulty: $DKU1$ and $IRS1$ in Table 5. If the components are faulty, this may explain the system behavior and the algorithm ends. At the same time, the communications of subsystems in Σ^- can be faulty. They are checked in Table 6.

c_i	$checkC(c_i)$	F_c	U_c
$DKU1$	1	$\{IRS1\}$	$\{DKU1\}$
$IRS1$	0	$\{IRS1\}$	$\{DKU1\}$

Table 5: Iterations of the $CheckComponents$ with $d_{protocol}$

Subsystems	checkCom	Partition
$\{MFD1, RA\}$	1	$p_{NAV.ARINC}$
$\{CMC1, CMC2, PMC1, PMC2\}$	1	$p_{NAV.ARINC}$

Table 6: Diagnostic results for subsystems

The $IRS1$ is not faulty, the algorithm is relaunched with $U_c = \{DKU1, IRS1\}$ and the other decomposition $d_{com} = \{p_{NAV.com1}, p_{NAV.com2}\}$. The algorithm iterations are described in Tables 7 and 8.

Once $checkP(p_{NAV.com2}) = 1$, we deduce that $MFD1$ is not faulty, see Table 7. At this step, the unfaulty components are $\{DKU1, IRS1, MFD1\}$, and the diagnosis is {RA}.

Here the RA is faulty with $p_{NAV.com1}$, and the algorithm ends. The solution is RA for $p_{NAV.com1}$. The data flow of the messages are checked as the impacted connections, wiring and, routing. The system specificities of the communication modeled with $com1$ five clues of the possible faults. Thanks to the impacted functionality, we know that only messages concerning the IRS roll are concerned. At this stage, the simulation of the message or the bad connection of the IRS are the two main solutions.

p_i	$checkP(p_i)$	U_c	F_c
$p_{NAV.com1}$	0	$\{DKU1, IRS1\}$	$\{RA, MFD1\}$
$p_{NAV.com2}$	1	$\{DKU1, IRS1, MFD1\}$	$\{RA\}$

Table 7: Iterations of $CheckMultiplicationPartition$ with d_{com}

Subsystems	checkCom	Partition
$\{RA, IRS1\}$	1	$p_{NAV.com1}$
$\{CMC1, DKU1\}$	1	$p_{NAV.com1}$
$\{CMC2, PMC1, PMC2\}$	1	$p_{NAV.com1}$

Table 8: Diagnostic results of subsystems with $p_{NAV.com1}$

4.2 Application with updated constraints

We describe a new problem: the navigation functionality and the performance function do not behave normally. The new constraint is $CONS = \{checkP(p_i), \ \forall \ p_i \in d_{NAV.protocol} \cup d_{NAV.com} \cup d_{PERF.protocol} \cup d_{PERF.com}\}$. The algorithm is loaded from $CheckMultiplicationPartition$ with the decomposition d_{com}. The algorithm iterations are described in Table 9. Once $checkP(p_{PERF.com2}) = 1$, we deduce that $CMC1$ is not faulty.We continue with $d_{protocol}$ knowing the $CMC1$ is not faulty in Table 10. We deduce that we have to check $DKU1$ and $CMC2$.

p_i	$checkP(p_i)$	U_c	F_c
$p_{PERF.com1}$	0	$\emptyset$	$\{CMC2\}$
$p_{PERF.com2}$	1	$\{CMC1\}$	$\{CMC2\}$

Table 9: Algorithm 2's iterations with d_{com}

p_i	$checkP(p_i)$	U_c	F_c
$p_{PERF.ARINC}$	0	$\{CMC1\}$	$\{DKU1, CMC2\}$
$p_{PERF.SERIAL}$	1	$\{CMC1\}$	$\{DKU1 CMC2\}$
$p_{PERF.MIL}$	0	$\{CMC1\}$	$\{DKU1, CMC2\}$

Table 10: Iterations of $CheckMultiplicationPartition$ with $d_{protocol}$

At this state, we check the components on the system. Since the reparation of $CMC2$ has fixed the problem, we conclude that $CMC2$ has been faulty. We also check the $DKU1$ configuration, and find nothing. The diagnosis is $\Delta = \{CMC2\}$.

The evolution of the number of faulty and unfaulty components is reviewed on figure 6. As expected, the number of unfaulty components is increasing with new tests, i.e tests

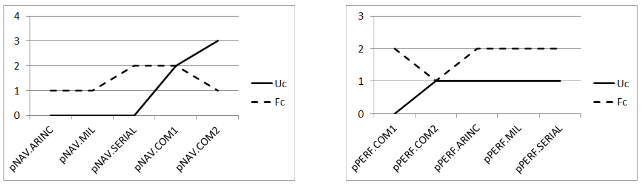


Figure 6: Evolution of the number of faulty and unfaulty components

of partitions. It reveals that the algorithm is converging to a solution because the number of components is limited.

5 Software implementation

5.1 Diagnostic software architecture

The algorithms are implemented in a spy software of ARINC and MIL-STD-1553 buses, see Figure 7. They are developed using C++ for effective diagnosis, and to be implemented in the AIRBUS software. The user interfaces are developed with Java 1.7 and the *Swing* Graphical User Interface (GUI) widget toolkit. The architecture of the diagnostic

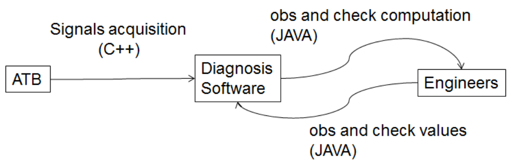


Figure 7: Data flow of the diagnosis software

framework has been adapted to the ATB specificities as described with the Model-View-Controller (MVC) paradigm on Figure 8. Three main objects are defined for the Model: the *Component*, the *Set*, and the *Partition* objects. Four main objects are defined in the View to define specific panels: the *diagnosisPanel*, the *constraintsPanel*, the *initialStatePanel* and the *resultsPanel* objects. The model is implemented with the *ArrayList* class. It is used to define the list of components, the subsystems and the list of partitions. eXtensible Markup Language (XML) files have been used to describe the system structure. The *Controller* dispatches the user requests and selects the panels for presentation. The diagnosis algorithm is implemented in it. A GUI is provided for handling user inputs such as partitions check values and components observations values.

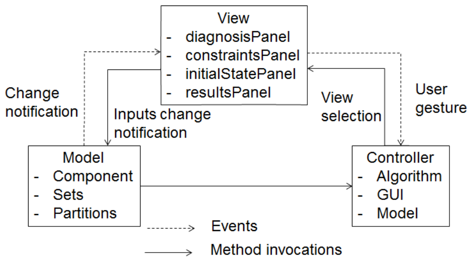
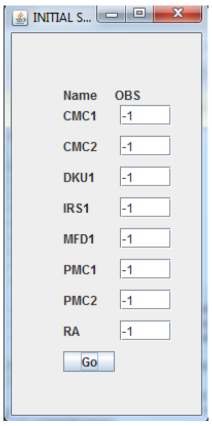


Figure 8: Architecture of the diagnosis software

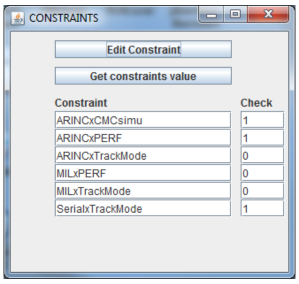
5.2 User interfaces

The panels are displayed one after the others for each step of the algorithm defined in the *Controller*. The



Name	OBS
CMC1	-1
CMC2	-1
DKU1	-1
IRS1	-1
MFD1	-1
PMC1	-1
PMC2	-1
RA	-1

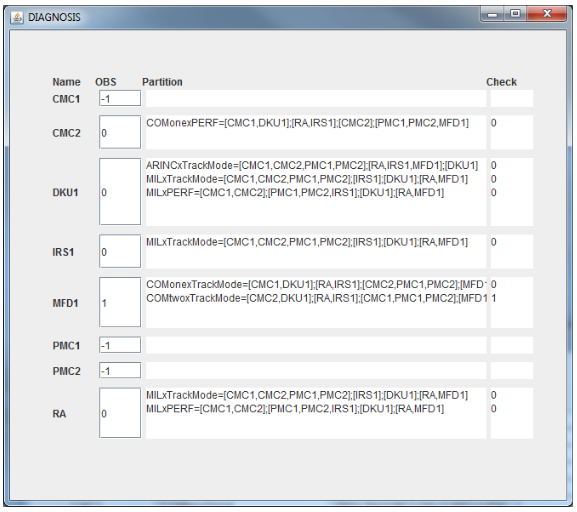
Figure 9: Initial state of the diagnosis



Constraint	Check
ARINCxCMCsimu	1
ARINCxPERF	1
ARINCxTrackMode	0
MILxPERF	0
MILxTrackMode	0
SerialxTrackMode	1

Figure 10: State of the constraints

initialStatePanel panel, Figure 9 defines the status of equipments before launching the diagnosis and a button the run the algorithm. The *check* values computed by the algorithm defined in the *Controller* are provided to the operator in Figure 11. The *constraintsPanel* panel lets to edit and update constraints, see Figure 10. The result of the diagnostic algorithm is provided on Figures 11. It gives the faulty components (observation equal to zero) and the impacted functionality. If a component is suspected, the data



Name	OBS	Partition	Check
CMC1	-1		
CMC2	0	COMonexPERF=[CMC1,DKU1];[RA,IRS1];[CMC2];[PMC1,PMC2,MFD1]	0

DKU1	0	ARINCxTrackMode=[CMC1,CMC2,PMC1,PMC2];[RA,IRS1,MFD1];[DKU1]
		MILxTrackMode=[CMC1,CMC2,PMC1,PMC2];[IRS1];[DKU1];[RA,MFD1]
		MILxPERF=[CMC1,CMC2];[PMC1,PMC2,IRS1];[DKU1];[RA,MFD1]
			0
			0
			0

IRS1	0	MILxTrackMode=[CMC1,CMC2,PMC1,PMC2];[IRS1];[DKU1];[RA,MFD1]	0

MFD1	1	COMonexTrackMode=[CMC1,DKU1];[RA,IRS1];[CMC2,PMC1,PMC2];[MFD
		COMtwoxTrackMode=[CMC2,DKU1];[RA,IRS1];[CMC1,PMC1,PMC2];[MFD1
			0
			1

PMC1	-1		
PMC2	-1		

RA	0	MILxTrackMode=[CMC1,CMC2,PMC1,PMC2];[IRS1];[DKU1];[RA,MFD1]
		MILxPERF=[CMC1,CMC2];[PMC1,PMC2,IRS1];[DKU1];[RA,MFD1]
			0
			0

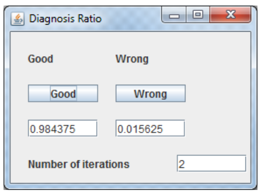
Figure 11: Diagnosis results

flow of the functional chain described by the partition must be checked. As described in the case study, it gives insights about the possible connections, wiring and, routing that can be wrong.

We compute the results $\Delta = \{ IRS1, DKU1, CMC2, RA \}$ and display them on Figure 11. If some components are unfaulty, we can update their status in Figure 9. The algorithm is relaunched using the "GO" button in Figure 9. The good diagnosis rate is evaluated on Figure 12. It is defined by the number of faulty components that the operator has to fix over the number of proposed faulty components.

5.3 Discussion

We have proposed a solution for the diagnosis of a complex system in aeronautics based on the MBD paradigm and the



Good Wrong

Good Wrong

0.984375 0.015625

Number of iterations 2

Figure 12: Good diagnosis rate

lattice concept. It is an other solution for the meta-diagnosis problem as described in [5] since we consider the test system environment as the main system. Belard has extended the framework, here we use the original one with the lattice concept to represent the system description. It is also provided a diagnostic algorithm implemented on the system to evaluate our method. Since hundreds of diagnosis are possible on the ATB, since it is not possible to check all those possibilities, we have introduced a methodology for the ATB diagnosis that reduce the number of iterations to get the diagnosis. We have upgraded the applications of MBD for avionics systems evaluated in [4] and [2]. It is proposed the integration and evaluation of a diagnostic algorithm for an ATB, taking the test systems environment into account. It differs from other applications of MBD like [8] because the model decomposition is driven by the test systems specificities that are represented with the lattice concept.

6 Conclusion

This paper extends the MBD approach to propose a diagnostic software that is developed for the diagnosis of test systems. The current framework is based on the lattice decomposition and is used to model a test system. First, the lattice decomposition has been used to decompose the system into its functionalities and connections. The second contribution consists in the proposal of an algorithm that reduce the diagnostic ambiguity. The lattice description has been implemented with JAVA native packages. The software architecture and diagnostic iterations are provided for a formal example and an industrial case study. The diagnostic algorithm has shown to reduce the number of faulty candidates. The results is either faulty equipment or a group of equipments with the associated system functionality that is unable to meet its goal. Together, they are sufficient to point out the reparations that will fix the system. The tests on the Avionics Test Systems in AIRBUS HELICOPTERS have shown good results. The development of models may confront our solution to many others real problems. In future works, algorithms will be improved with adaptable decompositions and automatic tests. Furthermore, as the method is generic, we want to demonstrate the validity of our method for others test systems used in AIRBUS HELICOPTERS.

References

[1] Canh Ly, Kwok Tom, Carl S. Byington, Romano Patrick, and George J. Vachtsevanos. Fault Diagnosis and Failure Prognosis for Engineering Systems: A Global Perspective. In *Proceedings of the Fifth Annual IEEE International Conference on Automation Science and Engineering*, CASE'09, pages 108–115, Piscataway, NJ, USA, 2009. IEEE Press.

[2] Arnaud Lefebvre, Zineb Simeu-Abazi, Jean-Pierre Derain, and Mathieu Glade. Diagnostic of the avionic equipment based on dynamic fault tree. In *Proceedings of the IFAC-CEA conference*, October 2007.

[3] Denis Berdjag, Jérôme Cieslak, and Ali Zolghadri. Fault detection and isolation of aircraft air data/inertial system. pages 317–332. EDP Sciences, 2013.

[4] Fabien Kuntz, Stéphanie Gaudan, Christian Sannino, Éric Laurent, Alain Griffault, and Gérald Point. Model-based diagnosis for avionics systems using minimal cuts. *DX 2011 22nd International Workshop on Principles of Diagnosis*, 2011.

[5] Nuno Belard, Yannick Pencole, and Michel Combacau. A theory of meta-diagnosis: reasoning about diagnostic systems. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, IJCAI'11, pages 731–737, Barcelona, Catalonia, Spain, 2011.

[6] Denis Berdjag, Vincent Cocquempot, Cyrille Christophe, Alexey Shumsky, and Alexey Zhirabok. Algebraic approach for model decomposition: Application for fault detection and isolation in discrete-event systems. *International Journal of Applied Mathematics and Computer Science (AMCS)*, 21(1):109–125, March 2011.

[7] Quang-Huy Giap, Stephane Ploix, and Jean-Marie Flaus. Managing Diagnosis Processes with Interactive Decompositions. In *Artificial Intelligence Applications and Innovations III*, IFIP International Federation for Information Processing, pages 407–415. 2009.

[8] Belarmino Pulido, Carlos Alonso-González, Anibal Bregon, Alberto Hernández Cerezo, and David Rubio. DXPCS: A software tool for consistency-based diagnosis of dynamic systems using Possible Conflicts. *25st Annual Workshop Proceedings, DX-14*, 2014.

[9] Veronique Delcroix, Mohamed-Amine Maalej, and Sylvain Piechowiak. Bayesian Networks versus Other Probabilistic Models for the Multiple Diagnosis of Large Devices. *International Journal on Artificial Intelligence Tools*, 16(3):417–433, 2007.

[10] Mattias Krysander, Jan Aslund, and Erik Frisk. A Structural Algorithm for Finding Testable Sub-models and Multiple Fault Isolability Analysis. *21st Annual Workshop Proceedings, DX-10*, 2010.

[11] Ronan Cossé, Denis Berdjag, David Duvivier, Sylvain Piechowiak, and Christian Gaurel. Meta-Diagnosis for a Special Class of Cyber-Physical Systems: the Avionics Test Benches. In *The 28th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, [Accepted]*, IEA/AIE 2015, Seoul, Corea, 2015.

[12] Johan de Kleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[13] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[14] Randall Davis and Walter C. Hamscher. Model-Based Reasoning: Troubleshooting. pages 297–346, July 1988. San Francisco, CA, USA.