IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 11, 2021, accepted March 11, 2021, date of publication March 17, 2021, date of current version March 23, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3066253

Self-Adaptative Troubleshooting for to Guide Resolution of Malfunctions in Aircraft Manufacturing

BELÉN RAMOS-GUTIÉRREZ[1], MARÍA TERESA GÓMEZ-LÓPEZ[1], DIANA BORREGO[1], RAFAEL CEBALLOS[1], RAFAEL M. GASCA[1], AND ANTONIO BAREA[2]

[1]Departamento de Lenguajes y Sistemas Informáticos, Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla, 41012 Sevilla, Spain
[2]Airbus Defense and Space San Pablo Sur, 41020 Sevilla, Spain

Corresponding author: Belén Ramos-Gutiérrez (brgutierrez@us.es)

This work was supported by the Project (RTI2018-094283-B-C33), funded by: FEDER/Ministry of Science and Innovation - State Research Agency.

ABSTRACT The increasing complexity of systems and the heterogeneous origin of the possible malfunctions bring about the necessity of redefining the troubleshooting processes. Troubleshooting comprises the set of steps for the systematic analysis of the symptoms after the detection of a malfunction. The complexity of certain systems, such as aircraft, means the origin of that malfunction can be any of several reasons, where diagnosis techniques support engineers in determining the reason for the unexpected behaviour. However, derived from the high number of components involved in an aircraft, the list of possible fault origins can be extremely long, and the analysis of every element on the list, until the element responsible is found, can be very time-consuming and error-prone. As an alternative, certain input/output signals can be read to prevent the substitution of a correctly functioning component, by validating its behaviour in an indirect way. In order to optimise the actions to perform, we have identified the relevant parts of the model to propose a troubleshooting process to ascertain the signals to read and the components to substitute, while striving to minimise the action cost in accordance with a combination of structural analysis, the probability of malfunction associated to the components, and the cost associated to each extra signal read and component substituted. The proposal has been validated in a system taken from a real scenario obtained in collaboration with the Airbus Defence and Space company. A statistical analysis of the degree of improvement of the troubleshooting process has also been included.

INDEX TERMS Decision-making process, model-based diagnosis, multi-objective function, troubleshooting.

I. INTRODUCTION

Fault diagnosis provides mechanisms to isolate the element responsible for a malfunction in accordance with a set of observations [1], [2]. However, signal values provided by monitoring systems are not always sufficient to isolate a suitable subset of possible components responsible for a malfunction, since a great number of root causes can be involved. Frequently, probability and cost are included in sorting the list of possible causes, although the number of possibilities can be extremely high. One option for the reduction of the number of candidates involves improving the observation of the system by developing a set of new readings of signals in the system [3], [4]. Troubleshooting is the reasoning process for the determination of what part of a system is causing a malfunction in accordance with a set of actions that guide the diagnoser to isolate the origin of misbehaviour. It is fundamental both to decrease the time taken to solve the problem and reduce the cost associated to reading signals of the system and to the component substitutions. Both aspects are especially important in contexts where there are numerous possible combinations of malfunctions. Aeronautic organisations can include a high number of components that are interconnected, such as the distribution of sensors to monitor

The associate editor coordinating the review of this manuscript and approving it for publication was Fatih Emre Boran.

the assembly process [5], and the Cyber-Physical Systems interaction with the rest of the software used in the company [6], [7] for troubleshooting and maintenance purposes [8].

The detection, isolation, and repair of components during aircraft assembly and in-service processes are very problematic [9], derived from the complexity of the systems, for both the number of components and the density of relations between them. The growing complexity in the systems and the heterogeneity of the failure nature, makes necessary to tackle the problem isolation improving the troubleshooting mechanisms [10]. During the aircraft assembly and in-service maintenance processes, several malfunctions might be found, which is the reason why a number of tests need to be applied to prevent the propagation of faults to other phases of the assembly. Both detection and resolution of possible malfunctions during the tests are fundamental in reducing the number of incidences before the aircraft is ready, thereby enhancing the product and process quality. Typically, the phases involved in the methodologies used in troubleshooting in systems such as aircraft include [11]: visual inspection, operational evaluation, problem classification, problem isolation, problem location, problem resolution, and a final operational evaluation to ensure the resolution of the problem. In other more specific cases, such as in-service support, the troubleshooting methodologies are complex and frequently, hand the responsibility of decision-making to the experts, instead of defining an ad-hoc set of steps for each case where both probability and cost are taken into account [12]. In complex contexts, such as aeronautic scenarios, for an observed malfunction there are several possible root causes, owing to the high connectivity between the components. This is why the visual or manual inspections, following a set of predefined steps, do not always constitute the optimal combination of activities to isolate the malfunction. This is why, it is considered interesting to create customised troubleshooting for each observed malfunction that has been adapted to the observations at each moment.

In this paper, we propose a framework to support the troubleshooting process after the detection of a malfunction in order to isolate the component responsible from a sorted list by automating the steps as much as possible, and guiding the operator minimising the cost of detection and that of the component substitutions. The method is based on the incorporation of the fault probability, substitution cost, the cost of the reading and the structural analysis, thereby creating a customised solution of each malfunction and sequence of signal. Since, for each element of the model, both probability and substitution cost must be combined, it is necessary to incorporate a function to optimise multiple objectives.

In order to validate our proposal, we have developed a solution that has been tested on the company Airbus Defence and Space. The concept of this proposal started as part of the context of the European project *Clean Sky 2*, and contributes to one of the tasks aligned with the key societal challenge for smart, green and integrated transport defined in Horizon 2020. The *Clean Sky 2* project built a full-scale in-flight demonstrator of innovative architectures and configurations to contribute toward advances in environmental and economic performance and to bring crucial competitiveness benefits to European industry. Through further initiatives, the final proposal was released and validated by the Airbus Defence and Space Manufacturing Engineering Transversal Systems department. A statistical analysis has been incorporated to demonstrate how the use of our proposal can reduce the troubleshooting effort.

The paper is organised as follows: Section II introduces the models of problems where our proposal can be applied. Section III presents the methodology proposed in this paper to be applied to the defined model. Section IV shows step by step, and in great detail, the different actions that would be carried out and the final result that is achieved with our proposal. Section V lays out the evaluation of our proposal and the provided advantages in a real scenario, as is the aircraft assembly process. Section VI shows the software architecture of our proposal. Section VII summarises the previous proposals in the area, and the reason why our solution implies a step forward in the literature. Finally, conclusions are drawn and future work is outlined.

II. TROUBLESHOOTING MODEL DESCRIPTION

The systems where troubleshooting techniques can be applied are those formed by a set of components interconnected by links. Both components and links can fail, but their main difference is that the links can be read to obtain information about the system behaviour. Those links can be automatically readable by computers (without any extra-cost) or through human action (implying cost and time). When an incorrect behaviour is triggered, it is necessary to find the elements responsible for the malfunction and to perform its substitution. In this section, the model of the systems supported by our proposal are formalised and an example is introduced which allows us to detail and explain the proposed methodology. Figure 1 shows a model, consisting of 18 components connected through 26 links, that represents a simplified model from a particular aircraft system. The same model can include various operational situations or configurations, called *operational modes* which perform specific operational scenarios, thereby activating a subset of their components depending on the mode.

Formalising these ideas, in our proposal, a **system model** involved in a troubleshooting process is described as the tuple $\langle E, L, OM, pf \rangle$, where E is a set of component elements, L represents the link elements that connect the component elements, OM is a set of operational modes where different parts of the systems are involved according to the operations executed in each case, and pf is the priority function which describes the importance between substitution cost, reading cost and malfunction probability of the elements to be sorted in the troubleshooting process. Each part of the system is detailed in the following subsections.

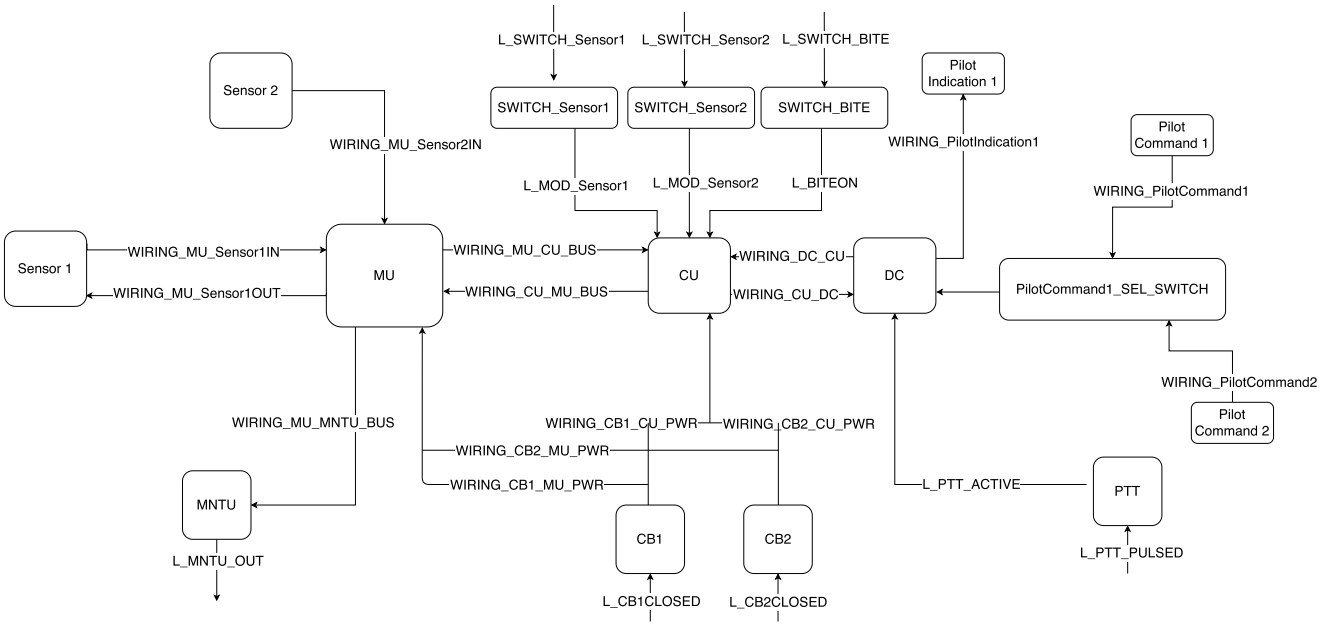

FIGURE 1. System example for troubleshooting.

A. COMPONENT ELEMENTS (E)

A system model consists of a set of component elements, each of which is described by an identifying name. The component elements of the example in Figure 1 are shown in Table 1.

TABLE 1. Component elements.

COMPONENT ELEMENT NAMES		
MU	CU	DC
PilotCommand1	PilotCommand2	PilotIndication1
CB1	CB2	Sensor1
MNTU	Sensor2	PTT
PilotCommand1_SEL_SWITCH	SWITCH_Sensor2	SWITCH_Sensor1
SWITCH_BITE	SWITCH_CB1	SWITCH_CB2

B. LINK ELEMENTS (L)

The links represent elements that connect component elements, and describe the inputs and outputs of the link. Each link is represented by the tuple ⟨*Name*: String, *Origin*: Component, *Destination*: Component, *Domain*: Integer/Real/Boolean, *Observability*: Boolean⟩. The *name* represents the link in a univocal way, *origin* is the component that produces the value that is assigned to a *variable* associated to the input or output of the link, *destination* is the component that receives the value of the variable transported by the link, and the *observability* is associated to the automatic readability of the link. The links can also fail, and therefore be substituted as a consequence of the detection of a malfunction, but the main difference with the components is that the values that are read in the links offer information regarding the complete behaviour of the element, since only one output is possible in a link.

According to the value of the *observability* (*true* or *false*) described in the previous tuple, the input or output variables of the links are defined as *known* or *readable* variables. In the model, the non-observable variables (i.e., internal to components) are not included since the components are presented as a black box.

- **Known variable**. When *observability* is *true*, the value of the variable associated to the link is known by the automatic monitorisation of the system, and its reading does not entail any extra effort, and the reading *cost* is equal to 0. Furthermore, in a particular way, if a link does not have an origin or destination component, then the *observability* of its associated variable is mandatorily *true* (it represents an input or output variable).
- **Readable variables**. When *observability* is *false*, the reading of the variables requires human intervention, thereby entailing an extra effort that requires labour time, which involves a reading *cost* greater than 0.

Link elements of the example in Figure 1 are shown in Table 2. In this example, links without origin correspond to input variables, and links without destination represent output signals.

C. OPERATIONAL MODE (OM)

Frequently, a single system can be utilised to support several different behaviours. Therefore, and depending on the specific operational mode in which the system is operating, on detecting a malfunction, it is not necessary to analyse all the elements of the system, but only those involved in that mode. This is especially important in the assembly process, where different parts of the system are involved in each test. An operational mode represents a state of the system model

TABLE 2. Link elements for the example.

NAME	ORIGIN	DESTINATION	DOMAIN	OBSERVABILITY
WIRING_MU_Sensor1IN	Sensor1	MU	*Boolean*	*False*
WIRING_MU_Sensor2IN	Sensor2	MU	*Boolean*	*False*
WIRING_CB1_MU_PWR	CB1	MU	*Boolean*	*False*
WIRING_CB2_MU_PWR	CB2	MU	*Boolean*	*False*
WIRING_CB1_CU_PWR	CB1	CU	*Boolean*	*False*
WIRING_CB2_CU_PWR	CB2	CU	*Boolean*	*False*
WIRING_CU_DC	CU	DC	*Boolean*	*False*
WIRING_DC_CU	DC	CU	*Boolean*	*False*
WIRING_PilotCommand2	PitloCommand2	DC	*Boolean*	*False*
L_PTT_ACTIVE	PTT	DC	*Boolean*	*False*
L_MOD_Sensor1	SWITCH_Sensor1	CU	*Boolean*	*False*
L_MOD_Sensor2	SWITCH_Sensor2	CU	*Boolean*	*False*
L_BITEON	SWITCH_BITE	CU	*Boolean*	*False*
WIRING_MU_CU_BUS	MU	CU	*Boolean*	*False*
WIRING_CU_MU_BUS	CU	MU	*Boolean*	*False*
WIRING_MU_MNTU_BUS	MU	MNTU	*Boolean*	*True*
WIRING_MU_Sensor1OUT	MU	Sensor1	*Boolean*	*True*
WIRING_PilotIndication1	DC	PilotIndication1	*Boolean*	*True*
WIRING_PilotCommand	PilotCommand1	DC	*Boolean*	*True*
L_CB1CLOSED		SWITCH_CB1	*Boolean*	*True*
L_CB2CLOSED		SWTICH_CB2	*Boolean*	*True*
L_SWITCH_Sensor2		SWITCH_Sensor2	*Boolean*	*True*
L_SWITCH_Sensor1		SWITCH_Sensor1	*Boolean*	*True*
L_SWITCH_BITE		SWITCH_BITE	*Boolean*	*True*
L_PTT_PULSED		PTT	*Boolean*	*True*
L_MNTU_OUT	MNTU		*Boolean*	*True*

where the system performs a specific operational scenario, for which the part of the model and a specific set of the components and link elements are active in each operation. It is described by the tuple $\langle name, E_{om}, L_{om}\rangle$ which represents the *name* of the operational mode, a subset of the components $E_{om} \subseteq E$, and a subset of the links $L_{om} \subseteq L$. Depending on the OM, different values for the probabilities of fault, reading cost, and substitution cost are assigned to each element (both components and links). Moreover, since the element links can be observed, they also have an observational cost that depends on the OM, derived from the difficulty to access a link according to the stage of the assembly. In summary, the E_{om} and L_{om} are described as the tuples: $E_{om} = \langle E, prob, sub\_cost\rangle$ and $L_{om} = \langle L, prob, sub\_cost, obs\_cost\rangle$.

Figure 2 details the part of the simplified aircraft system involved in the Operational Mode called *BITE*.

In order to ascertain whether the observed values of the known variables correspond to a correct behaviour, each OM must include a set of *Observations of Correct Behaviours*, described as follows.

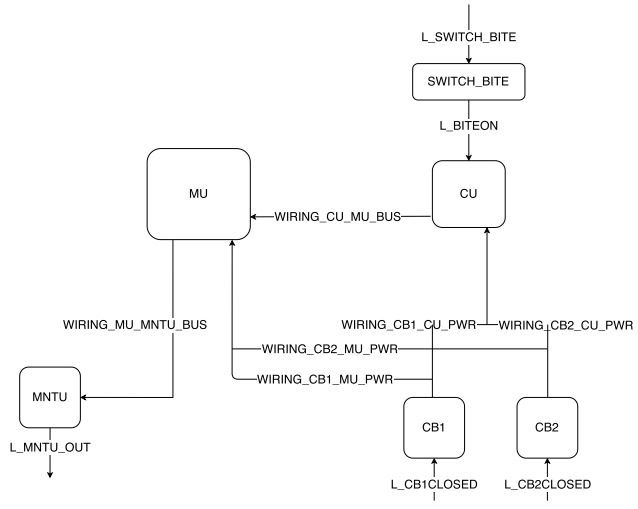

FIGURE 2. Simplified aircraft system in BITE operational mode.

1) OBSERVATIONS OF CORRECT BEHAVIOURS

Detecting correct or incorrect behaviour of a system can be carried out by observing the known variables. To this end, it is necessary to have previously determined the tuples of observations that represent correct behaviours. Each correct behaviour is identified by a tuple of values assigned to the known variables of the links in a system that does not present incorrect behaviour. When the known variables are not equal to any of the tuples that define correct behaviours, it is assumed that the observation corresponds to an incorrect behaviour.

Table 3 describes the *observations of correct behaviours* of the known variables for the BITE Operational Mode of the example. Since the domain of the variables for the example is *Boolean*, they can take value 0 or 1.

D. PRIORITY FUNCTION (PF)

As mentioned before, the selection of a component as being the element that is probably responsible for a malfunction derives from: a structural analysis of the relation of

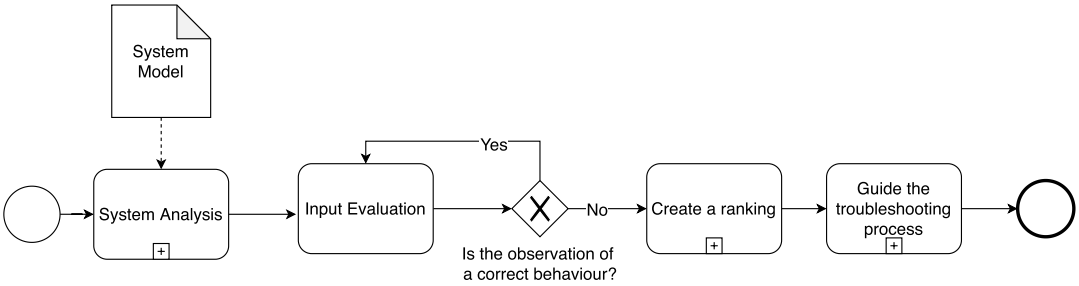

FIGURE 3. Steps of the methodology.

TABLE 3. Correct valuation for known variables in BITE operational mode.

KNOWN VARIABLES	CORRECT BEHAVIOUR
WIRING_MU_MNTU_BUS	1
L_CB1_CLOSED	1
L_CB2_CLOSED	1
L_SWITCH_BITE	1
L_PTT_PULSED	1
L_MNTU_OUT	1

the components (described by using the component and link elements), the probability of failure of the components (*probability*), and the costs of substitution and observation (*cost*). Once the involved components are determined by structural analysis, and in order to sort them into a ranking, a priority function is applied to each component by combining its *probability* and *costs* in a multi-objective function. The priority value obtained is the employed for sorting the components into order. One example of a priority function could be:

$$\sin(\frac{\pi}{2} \cdot E_{probability}) \cdot [0.5 \cdot E_{subsCost} + 0.5 \cdot E_{obsCost}]$$

where $E_{probability}$ is the probability of fault in the elements (both links and components) for the activated OM; $E_{subsCost}$ is the substitution cost of the element for the selected OM; and $E_{obsCost}$ is the observation cost. Furthermore, links have an associated observability cost (obs_cost), while, for components, $E_{obsCost}$ is the highest reading cost of all output links of the component. For this function, the domain of the results is in the range between 0 and 100. Therefore, for an element, a result close to 100 indicates that it can potentially be responsible for the failure, while if the result is close to 0, that it means that the element is almost definitely not responsible for the failure. Accordingly, if two elements obtain similar values from the priority function, it indicates that they both share roughly the same chance of being responsible for the failure. By applying the priority function to each element, a sorted ranking of elements is obtained, from highest to lowest probability. The proposed methodology explains how this interpretation can guide the engineer (Section III).

III. TROUBLESHOOTING PROCESS METHODOLOGY

Since the complexity of the systems can generate a very long list of possible root causes, it would be highly unprofitable to verify each and every possibility. This is the scenario where the operators must decide what is the best option, frequently following predefined guides designed by the engineers. To facilitate this process, our proposal customises the steps of the troubleshooting process according to the observations. This is why the proposed troubleshooting guide is needed to helping the decision making regarding on the most appropriate components to replace/analyse or the variables to read in order to obtain a more accurate and optimal diagnosis.

The methodology has four steps (as depicted in Figure 3): (1) system analysis; (2) system monitoring via input values; (3) creation of a ranking of elements according to the priority function; and (4) the execution of the guiding process to find the root of the fault, with a lower cost based on the highest probability.

The process starts a system analysis in accordance with the formalisation in Section II, which provides a structural analysis for the relations between the elements and defines how a malfunction affects the observations, as explained in Subsection III-A. In accordance with the observations at runtime, the system may be diagnosed, and a ranking is created that considers the probabilities of failure and their costs of substitution and observation by using the priority function (as explained in Subsection III-B). The troubleshooting guiding process enables the component responsible for the malfunction to be found while minimising the costs, as detailed in Subsection III-C.

In the following subsections, these steps are laid out in detail, and the example introduced in Section II is employed as an illustration.

A. SYSTEM ANALYSIS

In order to ascertain those elements which are potentially responsible for an observed malfunction (possible root cause), a structural analysis of the dependencies between the elements is necessary. The analysis process starts with the model of the system as input, which is decomposed (1) and analysed according to the dependencies between the components and links (2).

1) MODEL DECOMPOSITION

According to the formalisation, a component can have multiple inputs and outputs and therefore our proposal

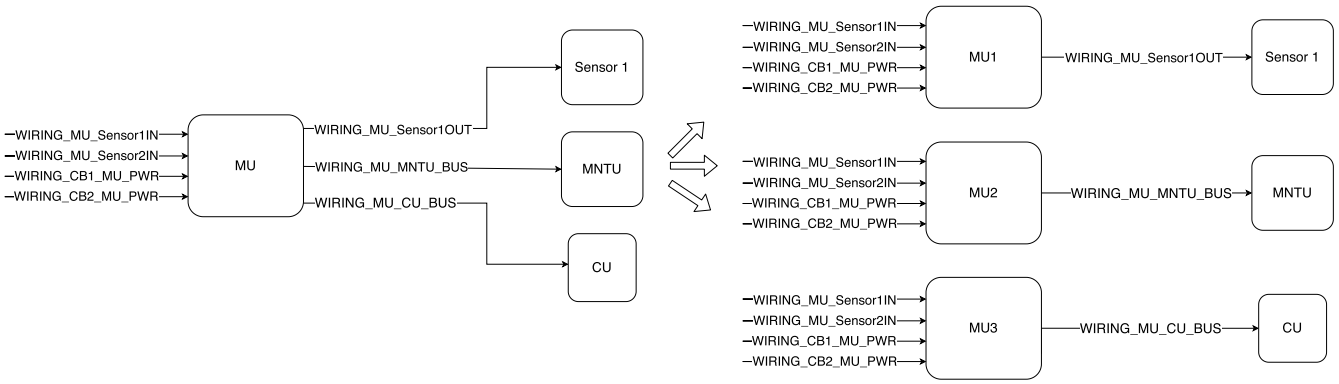

FIGURE 4. MIMO component MU transformed into three MISO components.

decomposes the system when necessary. Specifically, decomposition focuses on component elements with multiple outputs, the so-called MIMO (Multiple Input-Multiple Output) components. These MIMO components make exoneration tasks more complex, since a malfunction in a MIMO component can produce correct and incorrect outputs at the same time. Generally, in classic diagnosis, the correct observation of the output of a component exonerates it from the responsibility of a malfunction. However, if there are MIMO components in the model, this exemption becomes more complex during the troubleshooting process.

In order to apply exoneration according to a correct observation, we need to transform MIMO components into MISO (Multiple Input-Simple Output) components [13]. The transformation of a MIMO component E creates as many new MISO components as outputs of E, all of which contain the same input link elements as E. For the example, in Figure 1, there are 5 MIMO components: $\{DC, MU, CB1, CB2, CU\}$. In Figure 4, the component MU is transformed into three new components $MU1$, $MU2$, and $MU3$, one for each output of the MU original component.

With this decomposition, a MIMO component can only be exonerated if every single one of its MISO components are exonerated. Thus, situations like the following could occur: the output WIRING_MU_Sensor1IN is exonerated due to the correct reading of a variable. This leads to the MISO component $MU1$ being exonerated, but the entire MU component cannot be exonerated, since its other two outputs could still be incorrect. Subsequently if the MU component has to be analysed, then the analysis begins by reading its outputs that yet to be exonerated yet (that is, the exoneration of its remaining MISOs are sought).

2) DEPENDENCY ANALYSIS

There are many techniques that obtain the structural dependencies of a system [14]. This proposal focuses on the creation of a fault signature matrix [2], [15] that represents the set of variables that would be affected when each specific element fails. In the field of diagnosis, the use of fault signature matrices facilitates the isolation of components depending on the observations. For example, if the observation of a variable (output of an element E) denotes that E works correctly, then this implies that all the elements whose outputs are inputs of E also work correctly. These can therefore be removed from the list of possible root causes, thereby propagating the exoneration to related components.

After the model decomposition phase, the fault signature matrix is created to represent the relation between the element link variables, which can be observed in order to ascertain whether if the process is working correctly, and the elements of the system that can fail. Table 4 shows a part of the signature matrix for the system in BITE Operational Mode presented in Figure 2, to make the proposal understandable. The signature matrix includes only the MISO components involved in the specific operating model under analysis. Thus, for instance, components $MU1$ and $MU3$ do not appear, because they do not operate in this mode. Likewise, link elements that do not have an element as their origin never appear in the matrix, since if they were included, their column would always be empty, which means that under no circumstances reading them can help to exonerate any element.

In order to obtain the fault signature matrix [16], the structural relations in the system model are analysed using the information regarding its inputs and outputs. To enable the representation of which variables are affected when an element fails, a signature matrix has one row per element (component or link), and one column per known or readable variable of the system. When an X is located in a cell which is in the row of element E and the column of variable v, this means that if element E fails, then the value of v is affected by this fault. It is equivalent to the existence (in the model) of a path between the output of element E and the link variable v. However, the reverse implication is not possible, since an element E can work correctly while v is incorrect, due to another element implication. For the variable point of view, if v is incorrect, at least one of the elements with an X in its column must be working incorrectly. The signature matrix also includes the variables associated to the input value of the link element, such as L_BITEON_(I). As described above, a link element is an element with only one input and

TABLE 4. Signature matrix for BITE OM system.

ELEMENTS	LINK ELEMENT VARIABLES								
	L_BITEON	L_BITEON_(I)	WIRING_CB1 _CU_PWR	WIRING_CB2 _CU_PWR	WIRING_CU _MU_BUS	WIRING_CB1 _CU_PWR	WIRING_CB2 _MU_PWR	WIRING_MU _MNTU_BUS	L_MNTU_OUT
SWITCH_BITE	X	X						X	X
L_BITEON	X							X	X
CU					X			X	X
WIRING_CB1 _CU_PWR			X					X	X
WIRING_CB2 _CU_PWR				X				X	X
CB11			X		X			X	X
CB12						X		X	X
CB21				X	X			X	X
CB22							X	X	X
WIRING_MU _CU_BUS					X			X	X
MU2								X	X
WIRING_CB1 _MU_PWR						X		X	X
WIRING_CB2 _MU_PWR							X	X	X
WIRING_MU _MNTU_BUS								X	X
MNTU									X
L_MNTU_OUT									X

TABLE 5. Ranking of elements for an incorrect input tuple in BITE OM.

KNOWN VARIABLES	INCORRECT BEHAVIOUR
WIRING_MU_MNTU_BUS	**0**
L_CB1_CLOSED	1
L_CB2_CLOSED	1
L_SWITCH_BITE	1
L_PTT_PULSED	1
L_MNTU_OUT	**0**

(a) Tuple of incorrect values for known variables

ELEMENT	PRIORITY FUNCTION VALUE
WIRING_MU_MNTU_BUS	58.337
CB1	57.896
CB2	57.896
WIRING_CB1_MU_PWR	47.895
WIRING_CB2_MU_PWR	47.895
MU	8.916
MNTU	8.603

(b) Ranking of possible root cause elements

one output. The column of an input link element is equal to the column of the element link variables without an X in the element link. These variables are not included in the model since they are derived from the link element, and no such extra information is necessary.

B. RANKING CREATION

During the assembly and testing processes, the system functionalities are continuously monitored to verify the correct behaviour, to make a diagnosis if needed. If certain known variables present incorrect values, this means that the read input tuple does not match the operational mode. It therefore becomes necessary to formulate a hypothesis regarding the malfunction, and if the diagnosis activity begins. Depending on the specific known variables that are correct or incorrect, it is possible to isolate the potential elements involved in troubleshooting, since probably not all the elements that belong to the OM can be responsible for the detected malfunction. This will depend on the structural analysis (used to create the signature matrix) and according to the specific incorrect known variables. For example, Table 5(a) describes a tuple of known variable values that does not match with the correct behaviour in the OM, and therefore a diagnosis process should be executed. When an incorrect tuple is detected, the elements involved in the OM must be ordered, such as is shown in Table 5(b). The troubleshooting process starts analysing this list and the signature matrix.

It is important to note that MISO components never appear in the ranking, only MIMOs appear. This is because when a component has to be replaced, it must be completely replaced, since its division into MISOs is not physical, but logical. In this way, the MISOs intervene in the subsequent guiding process only to facilitate exemptions, while only the MIMO components appear in the ranking.

Furthermore, it is important to highlight that, as mentioned above, if two or more elements appear in the ranking with the same value for the priority function, then this indicates that they have the same value obtained using the priority function. In these circumstances, the operator can decide which is to be analysed first.

C. GUIDING THE TROUBLESHOOTING PROCESS

In classic approaches, and making use of a ranking of potential elements to be replaced, the troubleshooting starts by replacing the first element in that ranking (*WIRING_MU_MNTU_BUS* in the previous example), since it has the greatest value according to the priority function. Therefore, if the problem is solved, the diagnosis process is complete. In contrast, if the malfunction persists, the diagnosis process follows the order established in the ranking for the replacement of elements one by one until the origin of the malfunction is found.

However, due to the potential complexity of the models, the potential root causes can be very numerous and very

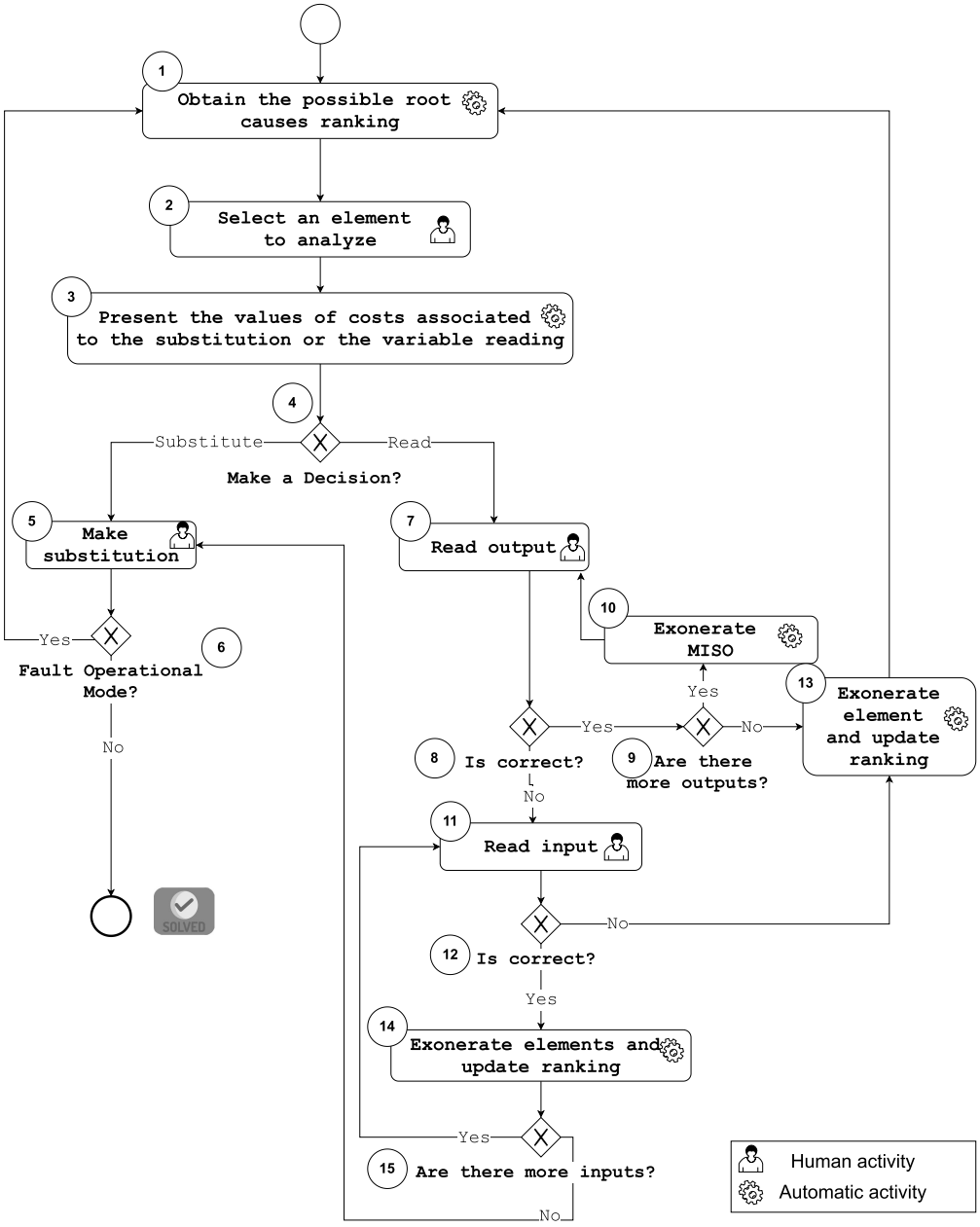

FIGURE 5. Stages of the guiding algorithm.

expensive to fix or replace, therefore, it may not be feasible to replace or repair the elements without being 100% certain that they are really causing the problem. That is why in the troubleshooting process, our proposal includes, the readings of readable variables to isolate the problem in an optimal way, and to prevent unnecessarily replacing correct elements whenever possible.

During the process, the steps presented schematically in Figure 5 are proposed as a guide, and they serve as support to explain, in greater detail, the different stages that make up the guiding process until the component responsible for the malfunction is detected.

According to the ranking **(Step 1)**, such as that shown in Table 5 (b), the operator must decide which element to analyse **(Step 2)**. The algorithm suggests analysing the elements in the order that they appear in the ranking; however, if the operator decides to analyse another element instead of that which the ranking order proposes, then the ignored element is relocated at the end of the list for future consideration. The exoneration or blame of the components can be determined in a direct (by means of their replacement) or an indirect way (reading readable variables) **(Step 3)**.

The information provided for each element to make the decision is composed of three types of costs:

- **Element substitution cost:** the cost of replacing the component/link element.
- **Cost of ascertaining whether the element is working correctly:** if the element is a component, then it might have one or more outputs, which are readable or known variables. In the calculation of this cost, we must differentiate between two scenarios: (1) the outputs of the component are correct, or (2) the component outputs are incorrect and all its inputs are correct. Consequently, this cost is equal to the sum of observation costs of the outputs plus the sum of the observation costs of the inputs. Similarly, with link elements, if the variable associated with the link is correct, then the cost of ascertaining that it is correct is equal to the cost of reading its variable. However, if the variable is incorrect, then the cost of knowing that the link element is correct is equal to the cost of reading its variable with an incorrect result plus the cost of reading the input, also with an incorrect result.
- **Cost of ascertaining whether the element is not working correctly:** This is the cost of finding out that its outputs are incorrect, plus the cost of reading all its inputs to verify that they are correct.

Based on this information, the operator has to decide (**Step 4**) between substituting the element or, if it is possible, analysing the element indirectly by reading variables. If the operator decides to substitute (**Step 5**), the element must be replaced and the persistence of the fault should be checked (**Step 6**). If the malfunction disappears, then the troubleshooting process ends since the responsible for the failure has been found. On the other hand, if the fault is still present, then the information regarding the performed substitution is recorded and the troubleshooting process continues once the ranking is updated accordingly.

For the example, it is decided to analyse the element by reading variables; the process starts reading the output of the element **(Step 7)** to exonerate it and the other elements related to that variable in the signature matrix (**Step 13**). If the variable is incorrect (**Step 8**) the inputs of the element must be read (**Step 11**) to ascertain whether they are incorrect or not (**Step 12**). The casuistry is as follows:

- If at least one input variable is incorrect, then the component under study can be exonerated since its incorrect output has been caused by another faulty component (**Step 13**).
- If an input is correct (**Step 15**), then the components related to it in the signature matrix can be exonerated (i.e., those that have an X in the column of the input variable).
- If every input is correct (**Step 16**), then it is implied that the component produces an incorrect output from correct inputs, and therefore it is possibly the component the possible responsible for the malfunction.
- If an output is correct (**Step 8**) and the element is MIMO, then only the MISO element can be exonerated **(Step 10)**, unless it is the last MISO component (from that MIMO element) that remains to be exempted, which would then exonerate the entire MIMO component, as well as those related to the output variable in the signature matrix **(Step 13)**.

However, it is important to note that although all readings are saved during execution, if an element is replaced, then these read variables are reset, since every time an element changes in the system, the system also changes and, therefore, previous readings are no longer reliable.

IV. GUIDING PROCESS FOR THE EXAMPLE

For a better understanding of the previous algorithm, an example of the trace of the troubleshooting is detailed below, where the element responsible for the malfunction is MU, although this is unknown before the troubleshooting process starts. Figure 6 shows the BITE OM diagram, which includes costs of the substitution of components and costs of reading variables, and has been labelled with the phases described in the algorithm to facilitate its understanding.

Details of the development of the guiding process for a specific scenario are now given, and include decisions, readings made, substitutions, etc.

1) Starting from the incorrect input tuple shown in Table 5(a), the ranking shown in Table 5(b) is obtained (**Step 1**). *WIRING_MU_MNTU_BUS* is the first element, a link, and the first candidate to be analysed according to the priority function results (**Step 2**). The system shows the three costs associated with the analysis of that element **(Step 3)**:
   - **Substitution cost**: 1,000.
   - **Cost of ascertaining whether *WIRING_MU_MNTU_BUS* is correct**:
     - **Best-case scenario**: 100. The cost of reading the output variable is correct, implies that the element is correct.
     - **Worst-case scenario**: 200. The cost of reading the output *WIRING_MU_MNTU_BUS* and input *WIRING_MU_MNTU_BUS_(I)* variables, to ascertain whether the component is working correctly (input and output incorrect) or incorrectly (input correct and output incorrect).
   - **Cost of ascertaining whether *WIRING_MU_MNTU_BUS* is incorrect**: 200. This cost is equal to ascertaining that it is correct in the worst-case scenario.

   Based on these three costs, the operator must decide whether to choose whether the replacement of the element or to perform the readings of a variable (**Step 4**). For the example, the decision is to read (**Step 7**), since it costs 200 instead of the 1,000 of the substitution. The algorithm then asks if the read variable is correct (**Step 8**). In this case, it is not correct (**Step 8**) and its input must be read (**Step 11**). The input of the link (*WIRING_MU_MNTU_BUS_(I)*) is incorrect (**Step 12**), and therefore, the link element is correct

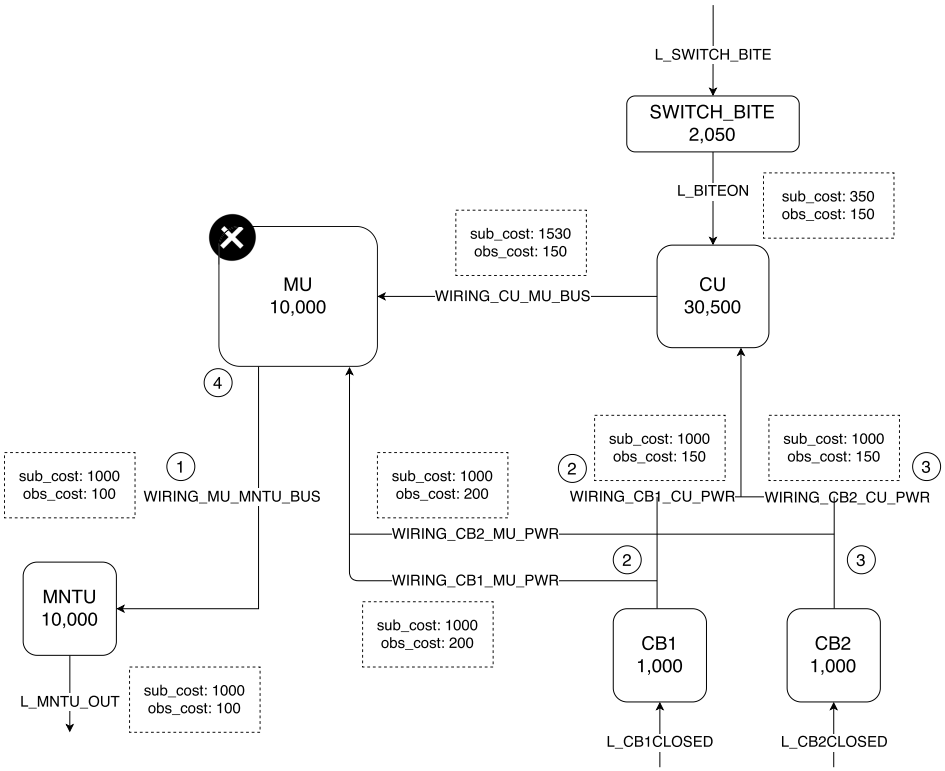

FIGURE 6. Troubleshooting example trace diagram.

TABLE 6. Ranking after first analysis.

ELEMENT	PRIORITY FUNCTION VALUE
CB1	57.896
CB2	57.896
WIRING_CB1_MU_PWR	47.895
WIRING_CB2_MU_PWR	47.895
MU	8.916
MNTU	8.603

and can be exonerated (**Step 13**). The ranking is then updated and the process continues (**Step 1**). Table 6 shows the new resulting ranking.

2) Following on with the example, the next element to be analysed is *CB1* (**Step 2**). The system shows the three costs associated to its analysis:
   - **Substitution cost**: 1,000.
   - **Cost of ascertaining whether *CB1* is correct**:
     - **Best-case scenario**: 350. This component is MIMO and is divided into two MISO components,(*CB11* and *CB12*). The cost of reading each output correctly (*WIRING_CB1_CU_PWR_(I)* and *WIRING_CB1_MU_PWR_(I)*) is 150 and 200 respectively, and the total cost their sum.
     - **Worst-case scenario**: 350. The worst-case scenario in ascertaining whether this component is correct is equal to the sum of reading its outputs, being incorrect, plus the reading of its input variable (*L_CB1_CLOSED*), being incorrect. Since *L_CB1_CLOSED* is a known variable, then its observation cost is 0 and the total cost is equal to 350.
   - **Cost of ascertaining whether *CB1* is incorrect**: 350. This cost is equal to the cost of ascertaining whether it is correct in the worst-case scenario.

   This time, when these costs are shown to the operator (**Step 3**), the option chosen is that of analysing by reading (**Step 4**), and the analysis begins by reading the outputs of element *CB1* (**Step 7**). These turn out to be correct (**Step 8**) and, therefore, the entire MIMO component can be exonerated, because both its MISO components have been exonerated (**Steps 10 and 13**). In addition, as result of the correct readings, two actions are carried out: (1) Elements are sought whose outputs are related to the correct variable (there is an X in the column of the variables *WIRING_CB1_CU_PWR_(I)* and *WIRING_CB1_MU_PWR_(I)*, and in the row of the potential components to be exonerated in the fault signature matrix), which, in this case, there are none; and (2) the correct reading of these variables is stored in the system so that it is taken into account in case they are needed in the future. However, the malfunctioning element has yet to be identified, and hence it is necessary to update the ranking and continue the search (**Step 1**). Table 7 shows the new ranking.
3) Now, the first element in the ranking is *CB2* (**Step 2**). The system displays the three related costs (**Step 3**):

TABLE 7. Ranking after the second analysis.

ELEMENT	PRIORITY FUNCTION VALUE
CB2	57.896
WIRING_CB1_MU_PWR	47.895
WIRING_CB2_MU_PWR	47.895
MU	8.916
MNTU	8.603

TABLE 8. Updated ranking after the third analysis.

ELEMENT	PRIORITY FUNCTION VALUE
WIRING_CB1_MU_PWR	47.895
WIRING_CB2_MU_PWR	47.895
MU	8.916
MNTU	8.603

- **Substitution cost**: 1,000.
- **Cost of ascertaining whether *CB2* is correct**:
  - **Best-case scenario**: 350. This component is MIMO and is divided into two MISO components (*CB21* and *CB22*). The cost of reading each output correctly(*WIRING_CB2_CU_PWR_(I)* and *WIRING_CB2_MU_PWR_(I)*), is 150 and 200 respectively.
  - **Worst-case scenario**: 350. The sum of reading its outputs (350), whereby at least one is incorrect, plus the reading of its input variable (*L_CB2_CLOSED*), which is also incorrect. Since *L_CB2_CLOSED* is a known variable, its observation cost is 0 and the total cost is equal to 350.
- **Cost of ascertaining whether *CB2* is incorrect**: 350. This cost is the same as the cost of the worst-case scenario.

By using this information, the operator chooses to analyse *CB2* by readings (**Step 4**), and the study begins by reading the its outputs (**Step 7**), which turn out to be correct (**Step 8**) and, therefore, the entire MIMO component can be exonerated, because both its MISO components have been exonerated (**Steps 10 and 13**). The two tasks that are executed after the correct readings are carried out in the same way as in the previous component. Finally, this is not the faulty component, and therefore it becomes necessary to update the ranking and continue the search (**Step 1**). Table 8 shows the new ranking.

4) *WIRING_CB1_MU_PWR* is the next element of the ranking to be analysed (**Step 2**). The system provides the following costs (**Step 3**):
   - **Substitution cost**: 1,000.
   - **Cost of ascertaining whether *WIRING_CB1_MU_PWR* is correct**:
     - **Best-case scenario**: 200. The cost of reading the output variable of the link that is correct.
     - **Worst-case scenario**: 200. The cost of reading the variable of the link that is incorrect plus the cost of reading its input variable (*WIRING_CB1_MU_PWR_(I)*), also incorrect. However, the reading of the variable *WIRING_CB1_MU_PWR_(I)*has already been carried out with a correct result and therefore it does not need to be read again.
   - **Cost of ascertaining whether *WIRING_CB1_MU_PWR* is incorrect**: 200. This cost is equal to the cost of reading the variable of the link, and ascertaining that it is incorrect, plus the cost of reading the variable that is the input of the link *WIRING_CB1_MU_PWR_(I)*, that is correct, making a total of 400. However, the reading of the variable *WIRING_CB1_MU_PWR_(I)* has already been carried out with a correct result, and therefore, it does not need to be read again, and no extra cost is incurred.

   Based on the information given, the operator must decide whether to choose to replace the element or to perform the readings of a variable (**Step 4**). The decision is to read (**Step 7**). Following on, the algorithm asks whether the read variable is correct (**Step 8**). In this case, it is correct and since it is a link, there are no more outputs (**Step 9**), and therefore the link element must be exonerated **(Step 13)**. In the new scenario, the fault persists, and therefore the ranking is updated and the process continues (**Step 1**).

TABLE 9. Updated ranking after the fifth analysis.

ELEMENT	PRIORITY FUNCTION VALUE
MU	8.916
MNTU	8.603

5) *WIRING_CB2_MU_PWR* is the next element of the ranking to be analysed (**Step 2**). Following the same reasoning as in the previous element, the system shows the following costs (**Step 3**):
   - **Substitution cost**: 1,000.
   - **Cost of ascertaining whether *WIRING_CB2_MU_PWR* is correct**:
     - **Best-case scenario**: 200.
     - **Worst-case scenario**: 200.
   - **Cost of ascertaining whether *WIRING_CB2_MU_PWR* is incorrect**: 200.

   The operator decides to analyse the link elements by readings again (**Step 7**). The reading is correct (**Step 8**) and since it is a link, there are no more outputs (**Step 9**), therefore, and the link element must also be exonerated (**Step 13**). The ranking is updated as shown in Table 9.

6) *MU* is the next item on the ranking to be tested (**Step 2**). The system provides the following costs (**Step 3**):
   - **Substitution cost**: 10,000.
   - **Cost of ascertaining whether *MU* is correct**:
     - **Best-case scenario**: 0. This component is MIMO, but in this mode, only one of its MISOs

is operative: *M2*, whose output is the variable *WIRING_MU_MNTU_BUS_(1)*. The reading cost of this variable is 100. Nevertheless it has been read previously, making the total cost for the MIMO element equal to 0.

  - **Worst-case scenario**: 150. This cost is equal to the cost of reading its output incorrectly, which is 0, plus the sum of the observation costs of its three inputs (*WIRING_CB1_MU_PWR, WIRING_CB2_MU_PWR, WIRING_CU_MU_BUS*), which must all be correct. Taking into account that the first two have already been exonerated, this cost is equal to the observation cost of *WIRING_CU_MU_BUS*.

- **Cost of ascertaining whether *MU* is incorrect**: 150. The cost is the same as for the worst-case finding that it is correct.

This time, again, the operator decides to analyse by reading variables (**Step 7**). As before, the process starts reading the output variables of the component, which, as previously analysed, presents an incorrect result **(Step 8)**. This leads to reading the inputs **(Step 11)**. This component has three inputs, two of which have already been read. When *WIRING_CU_MU_BUS* is verified, it is correct too, implying that the component is receiving correct information and producing erroneous information. Therefore, this component would be a potential cause of the failure, and the algorithm indicates to the operator that it should be replaced (**Step 5**). After the replacement, the fault disappears and the search process ends, since *MU* is responsible for the failure.

The convenience of applying our proposal instead of the classical proposals, where no partial readings are included, is laid out in the following section.

V. EVALUATION OF THE PROPOSAL

The effectiveness of our algorithm depends on the location in the ranking of the element responsible for the malfunction for each test. If the element responsible is the first one, then our algorithm incurs time and cost reading various variables. Otherwise, if the element responsible is deep within the list, our algorithm reduces the diagnosis cost drastically. To measure this effectiveness while taking into account the probability that an element appears in the different positions of the ranking, the troubleshooting guiding algorithm has been validated using a simplified aircraft system model of a real environment in Airbus Defence and Space factories. In order to verify the validity of the proposal, Subsection V-A shows how the algorithm has been tested in three different Operational Modes by simulating the possible element responsible for the malfunction for each mode and examining how the guidance of our algorithm could affect the decisions. This verification has been carried out to compare the benefits and disadvantages of our methodology versus the traditional methodology. Subsection V-B lists the conclusions drawn in the validation, the advantages of our approach, and those cases where its use can introduce major benefits.

A. VALIDATION OF THE PROPOSAL

In this section, our approach is tested on three different OMs from the same simplified aircraft system. This implies that, in each OM, certain elements are involved, while others are not, and therefore the lists of possible root causes change with each test. For illustrative reasons, and to be able to go into greater detail, give a full description of the tests for one OM, although the obtained results of the other two are also included in the paper. The details of the results of the other two tests are shown on the web.[1]

For every test, results are presented in accordance with two approaches: (1) classic, replacing elements in the order of the cost-probability relationship given in the ranking; and (2) our proposal, based on the reading of variables.

The tests are carried out, by using a defined ranking to determine the costs of discovering of the faulty element. These costs depend on the type of approach used and the location in the ranking of the element responsible. To gather every cost, both approaches and the positions of the element are studied for a later analysis. Table 10 contains the information regarding the results achieved for the OM BITE2. First, the ELEMENT column represents the ranking of possible root causes identified during the troubleshooting process, sorted in the order from highest to lowest according to the priority function. The column CLASSIC APPROACH contains the results of costs incurred in ascertaining whether an element is responsible for the failure or not, following the classic methodology of substituting elements according to the ranking order. In the classic approach, the cost of substituting the i-th component includes the cost of substituting every (i-1)-th previous element that turned out to be correct (the best-case or the worst-case scenario, depending on the case) and the element involved in each tuple. Regarding our approach, the set of columns PROPOSED APPROACH contains the same information with the exception that, in our proposal, the costs differ depending on whether a scanned element is working properly and its outputs are correct (lowest cost to ascertain whether it is correct) or reading every inputs and outputs, that must be performed when the output is incorrect and it is necessary to ascertain if the inputs are correct. When an incorrect element is detected, it is necessary to include the replacing cost to the cost of the column to determine the culprit. Moreover, both for correct and incorrect elements, the cost of analysing the element of the i-th row includes the cost of ascertaining that the (i-1)-th previous elements are also correct. Finally, the column SAVINGS contains the percentages of cost reduction of our proposal compared to the classic proposal in the analysis of each element (the percentage includes the substitution cost of each element plus the cost

[1] http://www.idea.us.es/ts2020/.

TABLE 10. Comparison of costs and steps of traditional and proposed solutions in BITE2 mode.

ELEMENTS	SUBSTITUTION COST	CLASSIC APPROACH	PROPOSED APPROACH		SAVINGS (%)
		Cost to know that is correct or not	Lowest Cost to know if it's correct	Greatest Cost to know if it's correct or culprit	
CB2	1,000	1,000	350	350	-35%
CB1	1,000	2,000	700	700	15%
WIRING_CB2_CU_PWR	1,000	3,000	850	850	38.33%
WIRING_CB1_CU_PWR	1,000	4,000	1,000	1,000	50%
WIRING_CU_MU_BUS	1,530	5,530	1,150	1,150	51.53%
CU	30,500	36,030	1,300	1,750	10.49%
MU	10,000	46,030	1,400	1,950	74.03%

TABLE 11. Costs per probability of failure and savings in BITE2 mode.

ELEMENTS	PROBABILITY	PROBABILITY in 100 tests	CLASSIC APPR. COST	PROPOSED APPR. COST
CB2	30	19.23	19,230.76	25,960.50
CB1	30	19.23	38,461.53	32,691
WIRING_CB2_CU_PWR	30	19.23	57,692.30	35,575.50
WIRING_CB1_CU_PWR	30	19.23	76,923.07	38,460
WIRING_CU_MU_BUS	20	12.85	70,897.43	34,438
CU	8	5.12	184,769.23	165,120
MU	8	5.12	236,051.28	61,184

to determine that this element is responsible, represented in the column with the highest costs in the proposed approach).

In the first test, we found a small ranking, in which only a few elements of the system intervene, as illustrated in Table 10. It can be observed that the costs are drastically reduced when the elements analysed are not responsible for the failure. When the defective element is at the top of the ranking, our proposal entails higher costs than the classic approach. However, the cost decreases significantly when the position of the defective item is lower in the ranking, and costs can be halved or reduced even more in certain cases.

If we focus on the column of Table 10 that shows the cost of determining whether the element is incorrect, the cost when each component fails can be ascertained. However, the analysis must include the number of times that the each element is actually the responsible for the malfunction. To evaluate our proposal, the probability information associated to each component is used. In the example, the probability is assessed in the range [0..100], where 0 represents the least probability of being responsible for the malfunction. In order to gain a more general idea of the advantages of applying our proposal, 100 tests are simulated, which take into account the different probabilities associated to each element. Table 11 includes the probability of each element where the summatory of the probability is an impossible 156 in this case. In order to obtain the column (Probability in 100 tests), a proportional adjustment is carried out. Based on the column probability in 100 test, we multiplied this value by the cost to solve the problem for each element, comparing the two approaches (columns Classic appr. cost and Proposed appr. cost). For the 100 elements, the total cost for the classic approach is $\sum_{i=1}^{Elements} probabilityIn100Tests_i * ClassicAppr.Cost_i = 684,025.64$, and $\sum_{i=1}^{Elements} probabilityIn100Tests_i * ProposedAppr.Cost_i = 393,717.94$, thereby saving 290, 307.69 €, which implies saving of 42.44% in the 100 tests.

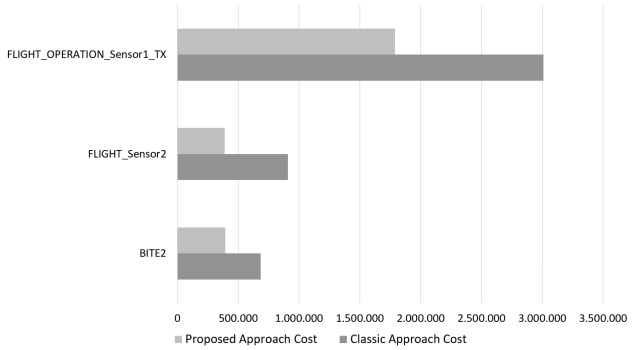

FIGURE 7. Total costs obtained for 100 tests with the two approaches.

B. CONCLUSIONS OF THE VALIDATION

Based on the observations collected in the previous subsection, four general behaviours can be concluded:

1) For elements in the first positions of the ranking, there are no major differences between the traditional solutions and our algorithm. In fact, costs remain the same or even increase using our approach, as well as the number of steps needed to reach the solution.
2) In the average cases, that is, those in which the elements are in the central positions of the ranking, the use of our proposal compared to the traditional approach enables costs to be reduced, and attains a greater reduction in the lower ranking positions.
3) For the last elements of the ranking, all costs are reduced by at least 50%. Those cases in which certain elements are extraordinarily expensive, should be borne in mind, since the costs of our proposal are also lower than the costs of the traditional methodology.
4) By carrying out a statistical analysis of the results, we have reached the conclusion that, assuming that each failure was repeated 100 times, and taking into

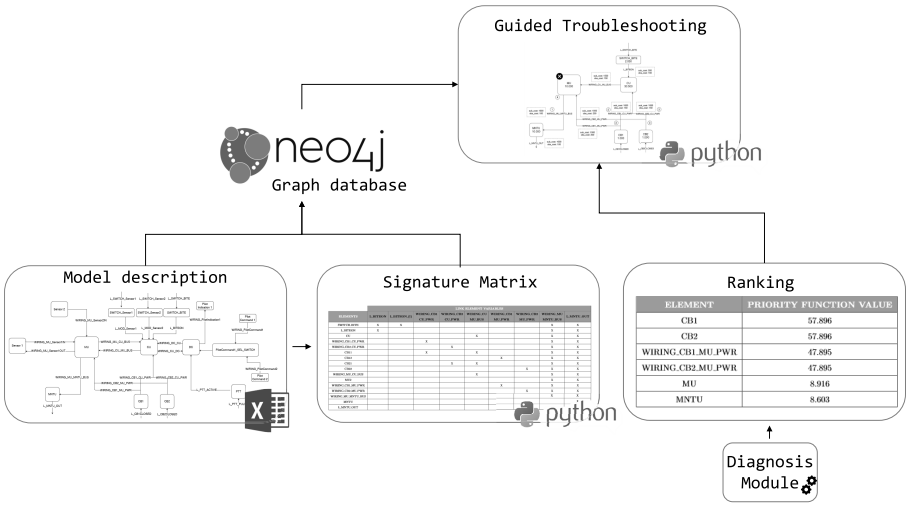

FIGURE 8. System architecture for guided troubleshooting.

account the probability of failure of each element, our proposal produces significant savings: 42.44%, 57.26%, and 40.54%. Figure 7 shows the large differences between the total costs for the approaches for each of the three tests.

VI. IMPLEMENTATION DETAILS

Since the solution is oriented towards guiding troubleshooting during the assembly processes, it is crucial that the implementation can be executed not only on computers, but also on tablets or other mobile devices. For this reason, the set of combined technologies is oriented towards the optimisation of time and resources, since the main aim is to achieve an efficient and fast response system, with a suitable performance and minimum resource consumption.

As shown in Figure 8, based on the description of the system model and the structural relationships between elements, the troubleshooting system needs a mechanism to describe the dependencies for the later isolation process. Hence, we propose the use of labelled graphs for modelling the problem representation, whereby it is possible to use a graph database to store and query these graphs, as well as storing the signature matrix that is used during the execution of the guided algorithm. On the other hand, the diagnosis module provides the ranking of possible root causes of failure, and, by using these possible causes, our system allows the operator to execute the guiding process to ascertain the real element responsible for the malfunction.

In order to achieve the aforementioned objectives, our graph database has been designed as follows:

- **Nodes**. We have used four different types of nodes: **elements**, **link variables**, **operational modes**, and **signature matrices**. For each node, the graph database only stores its name.
- **Edges**. Three types of edges are included:
  - *Operational-mode structural relationships*: These represent the structural relations that are established between the elements and variables of the system in the different operational modes. Thus, if, in a particular operational mode, a set of elements and variables must necessarily be connected and either active or inactive, then the edges linking them will appear in the graph. To represent this relationship, each edge that links an element and a variable has the name of an operational mode and contains additional information in the way of attributes, such as: (1) the type of relationship between the element and the variable (input, output, bidirectional); and (2) a value that represents the state in which the interface of the element must necessarily be (0: inactive; 1: active).
  - *Operational-mode costs and probabilities*: Each element has an edge to each operational mode. These edges specify their substitution cost and failure probability as an attribute. Similarly, each variable also has an edge to each operational mode, which contains the cost of observation of the variable as an attribute.
  - *Correspondence between the operational mode and its signature matrix*: These are edges that link the operational-mode nodes with the signature-matrix nodes. In this respect, it is possible to determine which operational mode is associated with which signature-matrix failure.

Figure 9 shows an example of a graph database of the BITE mode.

Our proposal is developed by using Python to create the logic of the algorithm described in Section III and by employing **Neo4j** as a graph database. **Neo4j** is a No-SQL graph-oriented database that is employed to store information

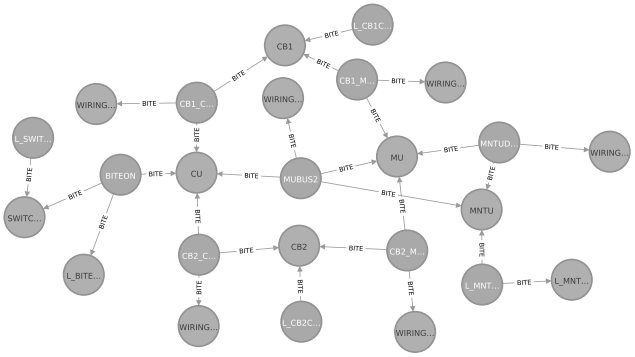

FIGURE 9. Example of a BITE mode graph.

related to the data model. Thanks to the structural relationships established between the elements of the studied subsystem, it can be modelled using a graph topology. For this type of information structuring, this type of database gives us significant advantages over an SQL solution, since the queries made on the data model imply adjacency relationships, which are as important as the data itself. In addition, due to the features of the algorithm, most of the queries require the navigation through different levels of depth, and this is always more efficient if graphs are used instead of SQL "*join*" operations. Moreover, it should be taken into account that for this small simplified problem, our database supports more than 100 nodes and 1, 600 edges.

On the other hand, the query language used by this database favours simplicity, which is an important fact to take into account, since the lighter the logic used by the system, the greater the chances of using it on different platforms and devices. Details of the implementation, more tests, and an example of the applicability of our proposal is available in the web.[2]

VII. RELATED WORK

In previous work, automated troubleshooting has brought significant benefits for quality assurance in manufacturing or network systems [17]. On these systems, failures can occur and various attempts could be made to repair them in the best way possible. Furthermore, on the aircraft manufacturing or similar complex systems, faults could propagate or affect numerous other systems, which leads to a great quantity of work to solve the fault. Our work has been motivated by the limitation of available information to perform the best fault troubleshooting process during the manufacturing. In these cases, many hypotheses have be to considered and the troubleshooting process could require a high number of steps to solve a determined failure. The selection of previous studies that are more closely related to our paper includes topics such as sequential fault diagnosis [18], interactive troubleshooting and troubleshooting using Artificial Intelligence techniques.

The MyAID application in [19] provide workers with interactive troubleshooting of industrial machines. It relies on hypermedia information systems, and shows step-by-step instructions using multimedia material. However, this study fails to specify how the application minimises the time and cost in the search for causes of failure.

Another set of studies related to fault diagnosis and troubleshooting for aircraft systems use Artificial Intelligence techniques. Reference [20] uses a system based on the combination of case-based reasoning and fault tree analysis, and provides a more precise fault diagnosis and obtains technical support for maintenance. The troubleshooting process uses the specific symptoms as input, and the output is a troubleshooting guide tree of similar cases identified. Another paper [21], presents an intelligent decision system for fault diagnosis of aircraft. The C4.5 Algorithm is employed to obtain the best decision trees from the training data available and Principal Component Analysis (PCA) to decrease the dimension of the input data. The results show high correct fault detection rates, low missed detection, and also obtains the false alarm rates. In this case it is necessary to make available a complete and proper dataset regarding the faults for the construction of a model for troubleshooting problems. However, this is not always possible. Other previous work proposes to optimising fault troubleshooting processes that obtain the highest efficacy. The efficacy could depend on various criteria, such as fault probability, cost, and time. The input of this work is a list of suspected components that have been identified as possible causes of failure, and the output is the optimal ordering of troubleshooting tasks to solve the fault in the system. For example, Liu [22] proposes a utility function based on the probability of components suspected of failing and the time required to verify each component in order to obtain the optimal ordering. Furthermore, a mathematical proof is given such that the ordering obtained minimises the mean troubleshooting time, cost, or a combination of the two. This work only takes into account the list of suspected components, while our work is richer because the variables associated with these components are also considered in order to optimise the troubleshooting process.

In reference [23], another approach towards effective troubleshooting decisions is proposed. It is based on Bayesian Networks and the Multi-criteria Decision Approach (MCDA). The efficacy of the troubleshooting process depends on fault probability, cost, time, and risk of repair action. The approach ensures a cost-saving, highly efficient, and low-risk troubleshooting selection in each step, where different alternatives are considered with regard to the previous criteria. An automobile engine startup failure is used as a case study, but fails to show any validation of the results obtained. In [24], the authors propose a framework to detect anomalies in aircraft systems during flight. However, it does not provide a systematic methodology for the resolution of the identified failures.

Recently, a set of new topics have appeared related to automated fault troubleshooting, such as Self-Healing [25], Smart Troubleshooting [10], and Smart Maintenance [26], [27]. These concepts include frameworks, methodologies and related tools. Furthermore, the modelling analysis, and

[2] http://www.idea.us.es/ts2020/.

recovery of information from failures is proposed in an automatic or (semi)automatic way. These could provide very interesting alternatives to addressing the problem of the effective fault troubleshooting process.

Finally, some other proposals have been developed to support both monitoring and troubleshooting using machine learning [28], [29], but they did not take into account the change of the observation in the model to reduce time and cost. However, to the best of our knowledge, our solution is the first to automatically create and adapt the troubleshooting problem according to observations, by optimising its solution taking into account the cost of signal reads and component substitutions.

VIII. CONCLUSION AND FUTURE WORK

A proper troubleshooting process, which minimises the time and cost of analysis in the search for causes of failure, is crucial for the daily operation of manufacturing companies. This is especially relevant with troubleshooting processes that work with systems whose components are expensive and whose complexity requires a very tricky analysis that can be extended widely in time, and can cause significant losses and delays. In this paper, we focus on the usefulness of reading variables and the interactivity of the troubleshooting guiding the process to improve its performance. Our solution is based on a graph-oriented approach that provides the user with the necessary information to help improve decision-making in the search for the element responsible for the failure. To this end, we have assumed the existence of 5 essential aspects: (1) a model that contains all the structural relationships between the different elements of the system, which can be interpreted as a graph; (2) the cost and probability associated with the failure of components, which will be used for ranking the causes ordered; (3) the ability to read values in intermediate elements (links); (4) the possibility of transforming a model with MIMO elements into MISOs; and, (5) the capability of being able to exonerate elements through observations. With all this information, it is possible to generate an algorithm capable of following a certain order, indicating the application of actions that can be carried out on each element of the ranking, and the costs that his/her decision entails, as well as acting accordingly, exonerating, blaming or isolating the different elements and variables that explain the malfunctioning of the system. The algorithm has been tested in a real scenario, and the degree of cost reduction that can be achieved is analysed.

Although, in some cases, our solution may seem more tedious than the traditional methodology, our validation and the fact that the experts participate in the project demonstrate that costs are reduced and that the unnecessary substitution of very expensive elements may easily disappear.

As an extension to this paper, we continue to work on improving the ranking order to ensure the optimal way to analyse the system. Furthermore, we are investigating how to include 3 types of multiple failure: more than one single simultaneous failure; failures where there is more than one element responsible for the fault; and failures that produce multiple variables with incorrect states. Consequently, we are also analysing the best way to include information from this multiplicity in the ranking prioritisation function. Finally, we are working on the analysis of decision-making and results obtained in each execution, using machine learning, in an effort to improve the weights given for probability in an automated way and based on real data.

REFERENCES

[1] M. Cordier, P. Dague, M. Dumas, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès, "A comparative analysis of AI and control theory approaches to model-based diagnosis," in *ECAI 2000*, W. Horn, Ed. Berlin, Germany: IOS Press, Aug. 2000, pp. 136–140.

[2] R. Ceballos, M. T. Gómez-López, R. M. Gasca, and C. D. Valle, "A compiled model for faults diagnosis based on different techniques," *AI Commun.*, vol. 20, no. 1, pp. 7–16, 2007.

[3] D. Borrego, M. T. Gómez-López, and R. M. Gasca, "Minimizing test-point allocation to improve diagnosability in business process models," *J. Syst. Softw.*, vol. 86, no. 11, pp. 2725–2741, Nov. 2013.

[4] L. Travé-Massuyès, T. Escobet, and R. Milne, "Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem," in *Proc. 17th Int. Joint Conf. Artif. Intell. (IJCAI)*, B. Nebel, Ed. Seattle, WA, USA: Morgan Kaufmann, 2001, pp. 551–556.

[5] S. Du and L. Xi, "Fault diagnosis in assembly processes based on engineering-driven rules and PSOSAEN algorithm," *Comput. Ind. Eng.*, vol. 60, no. 1, pp. 77–88, Feb. 2011.

[6] E. Frontoni, J. Loncarski, R. Pierdicca, M. Bernardini, and M. Sasso, "Cyber physical systems for industry 4.0: Towards real time virtual reality in smart manufacturing," in *Augmented Reality, Virtual Reality, and Computer Graphics*, L. T. D. Paolis and P. Bourdot, Eds. Cham, Switzerland: Springer, 2018, pp. 422–434.

[7] A. Tedesco, M. Gallo, and A. Tufano, "A preliminary discussion of measurement and networking issues in cyber physical systems for industrial manufacturing," in *Proc. IEEE Int. Workshop Meas. Netw. (M N)*, Sep. 2017, pp. 1–6.

[8] U. Wetzker, I. Splitt, M. Zimmerling, C. A. Boano, and K. Römer, "Troubleshooting wireless coexistence problems in the industrial Internet of Things," in *Proc. IEEE Int. Conf. Comput. Sci. Eng. IEEE Int. Conf. Embedded Ubiquitous Comput., 15th Int. Symp. Distrib. Comput. Appl. Bus. Eng.*, Paris, France: IEEE Computer Society, Aug. 2016, p. 98.

[9] H. Warnquist, J. Kvarnström, and P. Doherty, "A modeling framework for troubleshooting automotive systems," *Appl. Artif. Intell.*, vol. 30, no. 3, pp. 257–296, Mar. 2016.

[10] M. Caporuscio, F. Flammini, J. Thornadtsson, N. Khakpour, and P. Singh, "Smart-troubleshooting connected devices: Concept, challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 111, pp. 681–697, Oct. 2020.

[11] N. Papakostas, P. Papachatzakis, V. Xanthakis, D. Mourtzis, and G. Chryssolouris, "An approach to operational aircraft maintenance planning," *Decis. Support Syst.*, vol. 48, no. 4, pp. 604–612, Mar. 2010.

[12] R. Kannan, S. S. Manohar, and M. S. Kumaran, "Nominal features-based class specific learning model for fault diagnosis in industrial applications," *Comput. Ind. Eng.*, vol. 116, pp. 163–177, Feb. 2018.

[13] Y. Jiang, B. An, M. Huo, and S. Yin, "Design approach to MIMO diagnostic observer and its application to fault detection," in *Proc. IECON-44th Annu. Conf. IEEE Ind. Electron. Soc.*, Washington, DC, USA, Oct. 2018, pp. 5377–5382.

[14] M. Blanke, M. Kinnaert, J. Lunze, M. Staroswiecki, and J. Schrder, *Diagnosis and Fault-Tolerant Control*, 2nd ed. Berlin, Germany: Springer, 2010.

[15] J. C. Chan and J. A. Abraham, "A study of faulty signatures using a matrix formulation," in *Proc. Int. Test Conf.*, Sep. 1990, pp. 553–561.

[16] D. Jung, "A generalized fault isolability matrix for improved fault diagnosability analysis," in *Proc. 3rd Conf. Control Fault-Tolerant Syst. (SysTol)*, Barcelona, Spain, Sep. 2016, pp. 519–524.

[17] T. Ogata, A. Takeuchi, S. Fukuda, T. Yamada, T. Ochi, K. Inoue, and J. Ota, "Characteristics of skilled and unskilled system engineers in troubleshooting for network systems," *IEEE Access*, vol. 8, pp. 80779–80791, 2020.

[18] M. Vomlelová and J. Vomlel, "Troubleshooting: NP-hardness and solution methods," *Soft Comput.—Fusion Found., Methodologies Appl.*, vol. 7, no. 5, pp. 357–368, Apr. 2003.

[19] V. Villani, N. Battilani, G. Lotti, and C. Fantuzzi, "Myaid: A troubleshooting application for supporting human operators in industrial environment," *IFAC-PapersOnLine*, vol. 49, no. 19, pp. 391–396, 2016.
[20] X. P. Yu, Q. Li, and X. Hu, "Aircraft fault diagnosis system research based on the combination of CBR and FTA," in *Proc. 1st Int. Conf. Rel. Syst. Eng. (ICRSE)*, Oct. 2015, pp. 1–6.
[21] S. A. Z. Wang, J.-L. Zarader, and K. Youssef, "A decision system for aircraft faults diagnosis based on classification trees and PCA," in *Intelligent Autonomous Systems 12*. Berlin, Germany: Springer, 2013, pp. 411–422.
[22] J. Liu, "Optimal task ordering for troubleshooting systems faults," in *Proc. IEEE Aerosp. Conf.*, Mar. 2005, pp. 3709–3714.
[23] Y. Huang, Y. Wang, and R. Zhang, "Fault troubleshooting using Bayesian network and multicriteria decision analysis," *Adv. Mech. Eng.*, vol. 6, Jan. 2014, Art. no. 282013.
[24] H. Lee, G. Li, A. Rai, and A. Chattopadhyay, "Real-time anomaly detection framework using a support vector regression for the safety monitoring of commercial aircraft," *Adv. Eng. Informat.*, vol. 44, Apr. 2020, Art. no. 101071.
[25] A. Asghar, H. Farooq, and A. Imran, "Self-healing in emerging cellular networks: Review, challenges, and research directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1682–1709, 3rd Quart., 2018.
[26] Y. Liu, Y. Wu, and Z. Kalbarczyk, "Smart maintenance via dynamic fault tree analysis: A case study on Singapore MRT system," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 511–518.
[27] M. Ashjaei and M. Bengtsson, "Enhancing smart maintenance management using fog computing technology," in *Proc. IEEE Int. Conf. Ind. Eng. Eng. Manage. (IEEM)*, Dec. 2017, pp. 1561–1565.
[28] A. D'Alconzo, P. Barlet-Ros, K. Fukuda, and D. Choffnes, "Machine learning, data mining and big data frameworks for network monitoring and troubleshooting," *Comput. Netw.*, vol. 107, pp. 1–4, Oct. 2016.
[29] C. Zhang, Y. He, B. Du, L. Yuan, B. Li, and S. Jiang, "Transformer fault diagnosis method using IoT based monitoring system and ensemble machine learning," *Future Gener. Comput. Syst.*, vol. 108, pp. 533–545, Jul. 2020.

BELÉN RAMOS-GUTIÉRREZ is currently a Software Engineering and Technology Ph.D. Student with the University of Seville. Her research interests include process mining, data extraction, and optimisation for process mining techniques; and troubleshooting and decision support systems in industrial environments. She is also working as a Predoctoral Researcher with the University of Seville and collaborate on projects involving industrial aspects related to logistics-port and aeronautical environments. Her goal is to improve and automate the extraction and processing of heterogeneous data from industrial environments for exploitation with process mining techniques.

MARÍA TERESA GÓMEZ-LÓPEZ is currently pursuing the Ph.D. degree in computer science. She is also a Lecturer with the University of Seville and the Head of the IDEA Research Group. Her research interests include model-based diagnosis in business processes and data management in big data environment. She has led several private and public research projects and has published more than 20 impact articles (DSS, IS, DKE, IST, and so on). She was nominated as a member of several Program Committees (BPM, ER, EDOC, CAISE Doctoral Consortium, and so on). She has been reviewing for several international journals. She has been invited speaker at various conferences and summer schools.

DIANA BORREGO received the Ph.D. degree in computer science. She is currently a Lecturer with the University of Seville and a member of the IDEA Research Group. Her research interests include the verification and diagnosis of business processes through automatic reasoning for their quality improvement. Her works have appeared in international conferences and journals, including *Information and Software Technology*, *Data & Knowledge Engineering*, the *Journal of Systems and Software*, and *Computers in Industry*.

RAFAEL CEBALLOS received the M.Sc. and Ph.D. degrees from the Department Computer Languages and Systems, University of Sevilla, in 2002 and 2011, respectively. He is currently working as an Assistant Professor with the University of Sevilla. He is the author and coauthor of many book chapters, conference papers, and impact journals articles (*Applied Sciences*, *Information and Software Technology*, and *Data & Knowledge Engineering*). He has been awarded in CAEPIA-05 Doctoral Consortium. He was an Invited Speaker with the International Summer School on Fault Diagnosis of Complex Systems in 2019. His research interests include business processes and data management, model-based diagnosis, software testing, and cybersecurity.

RAFAEL M. GASCA received the Ph.D. degree in computer science, in 1998. He is currently a Professor with the University of Seville, Spain, where he has been responsible for the Quivir Research Group since 1999. His research interests include fault diagnosis, cybersecurity technologies, and business process management systems. He has been involved in European and Spanish research projects and has published numerous articles in *Computer Science* journals and international conferences.

ANTONIO BAREA has been working with Airbus since 2007. He is currently the responsible for the Avionics & Mission Systems Technologies Industrialization as a part of the Manufacturing Engineering. He has led many industrial innovations to improve the quality and reduction of costs for system testing process in the aerospace industrial facilities, which are being published or under patent review, such as the Mini Multi Interface Box Simulator (MMIBS), the Prow Radar Obstacle Simulator for Testing (PROST), Flight Recorder Industrial & Integral Data Analyzer (FRIIDA), and the latest, the Automatic Troubleshooting System (ATS).