Multilayer Architecture for Fault Diagnosis of Embedded Systems

Daniel N. Maas[1], Renan Sebem[2], André B. Leal[3]

[1] *LiDAR Software Architecture team at Bosch Braga, Portugal*

[2,3] *Group of Systems Automation and Robotic (GASR), Department of Electrical Engineering, Santa Catarina State University, Joinville, Brazil*
andre.leal@udesc.br

ABSTRACT

This work presents a multilayer architecture for fault diagnosis in embedded systems based on formal modeling of Discrete Event Systems (DES). Most works on diagnosis of DES focus in faults of actuators, which are the devices subject to intensive wear in industry. However, embedded systems are commonly subject to cost reduction, which may increase the probability of faults in the electronic hardware. Further, in tech support of a product without a diagnosis system, it takes time to identify if the fault is in software or the electronic board. In this case, the most common solution is to replace the whole electronic board with an updated version of the code. We propose a modeling approach which includes the isolation of the source of the fault in the model, regarding three layers of embedded systems: software, hardware and sensors & actuators. The proposed method is applied to a home appliance refrigerator and after exhaustive practical tests with forced fault occurrences, all faults were diagnosed, precisely identifying the layer and the faulty component. The solution was then incorporated into the product manufactured in industrial scale.

1. INTRODUCTION

The reliability and safety of engineering systems are hot topics of research. These qualities are indispensable in any engineering system, and the demand on them has an ever-increase trendline. Faults may compromise the reliability and safety of these systems, and therefore the possibility of avoiding a fault is a desirable feature. The study of faults in engineering systems comprehend many different frameworks, such as diagnosis and prognosis for continuous, hybrid and Discrete Event Systems (DES) (Vignolles, Chanthery, & Ribot, 2020); Remaining Useful Life (RUL) estimation (Ammour, Leclercq, Sanlaville, & Lefebvre, 2017); Prognosis & Health Management (PHM) (Goebel & Rajamani, 2021).

Fault diagnosis of engineering systems has many methods for different kind of systems. Even when narrowing down to techniques of embedded systems diagnosis there are many different approaches. Many of these approaches are based in complex mathematical backgrounds such as stochastic models (Ge, Nakajima, & Pantel, 2015), signal processing techniques (Bennouna & Roux, 2013; Lu, He, & Zhao, 2018; Lu et al., 2020) and hybrid models (Pons, Subias, & Trave-Massuyes, 2015). Such approaches may be out of reach for a software designer and, in this sense, we take advantage of the DES' formalism for fault diagnosis to develop a method of fault diagnosis for embedded systems.

Discrete Event Systems are dynamical systems with a discrete state space, in which the state-transition, i.e. the evolution of the system, is event-driven (Cassandras & Lafortune, 2008). Many kinds of systems are adequately represented by discrete event models, such as embedded systems; manufacturing systems; computer networks; heating, ventilation, and air conditioning systems (HVAC systems); and traffic control systems. The framework of fault diagnosis & prognosis of DES may be applied in all of these systems, which would improve its safety and reliability. DES can be modeled as automata & languages, which is a branch of the theory of computation, and therefore is well known to a software designer. In DES, fault diagnosis (Zaytoon & Lafortune, 2013) and fault prognosis (Watanabe, Sebem, Leal, & Hounsell, 2021) have different conditions. Roughly speaking, the conditions for prognosis are stronger (i.e. more difficult to reach) than the conditions for fault diagnosis, which means that, to fulfill the prognosability condition it is necessary that, from the observable behavior of the system, it is possible to infer about future fault occurrences, while that in diagnosis the detection of the fault is performed after its occurrence (Watanabe, Leal, Cury,

https://doi.org/10.36001/IJPHM.2021.v12i2.3067

& de Queiroz, 2021). In other words, a prognosable system is always diagnosable, however, a diagnosable system is not necessarily prognosable. In this paper we are interested in diagnosis of DES, as it has many formal frameworks such as *active diagnosis* (Moreira & Leal, 2020), *fault tolerant control* (Fritz & Zhang, 2018), and *diagnosis of sensor failures* (Takai, 2021), which aid the development of new strategies.

The process of fault diagnosis is formed by three main tasks: detection, isolation and identification (Zaytoon & Lafortune, 2013). Fault detection is recognizing a malfunction within the system. Fault isolation is determining which component has caused the fault. Finally, fault identification is detailing characteristics of the faults nature (e.g. a relay stuck closed).

In fault diagnosis of DES, these three tasks are not clearly defined, as it depends on the modeling abstraction taken by the designer. In other words, depending on the level of abstraction of the model, only fault detection is possible, requiring another techniques for isolation/identification. In order to isolate the fault, each component which is subject to fault should be modeled.

After a fault occurrence it is important to take an automatic recovery action or simply externalize this fault information (e.g. in a display) to be used during a corrective maintenance intervention. Thus, the more accurate is the isolation of the fault, the more effective will be the recovery action or the corrective maintenance.

To illustrate this, let us take an example in which a pump is used to drain the water from inside to outside of a washing machine. Consider a fault which leads to the impossibility of emptying the water from the basket, thus, the sources of faults could be:

a) Software – an error or inconsistency in the pump control algorithm, due to an unknown bug, or even due to an electromagnetic interference that can corrupt data coming from the system, thus preventing the pump from starting;

b) Hardware – a fault in the electronic board, such as a damaged component or circuit, preventing the pump from starting;

c) Actuator – a mechanical/electrical fault in the pump.

In most diagnosis approaches, this fault would simply be modeled as a fault in the pump (actuator) (Naha, Thammayyabbabu, Samanta, Routray, & Deb, 2017; Lu et al., 2018, 2020; Ning, Han, Wu, & Wang, 2018; Lu, He, Yuan, & Kong, 2017; Gandhi, Turk, & Dahiya, 2020; Ranade, Provan, El-Din Mady, & O'Sullivan, 2020; Guo et al., 2019). In industrial environments, this modeling approach is adequate because the actuators are subject to intensive wear in unfavorable conditions, such as high temperature, dust, etc. Also, the industrial electronics are robust and oversized, i.e they have high reliability and the probability of fault in electronic components are low. Finally, considering software faults in industrial environments, it is common that the software developer is always accessible, which may quickly intervene to solve the problem (different of an electronic product which would be sent to the tech support). Also, in industrial environments there are software tools which allows the supervision of the system, with logs and alarms (such as Supervisory Control and Data Acquisition – SCADA), facilitating the identification of a fault in the software.

In embedded systems manufactured in industrial scale, cost reduction and time to market are necessary factors to maintain the business competitiveness. These factors may increase the probability of faults in the software and electronic hardware. Moreover, when a bug in the software shows up, the consumer will send the product to the tech support. In this case, the most common solution is to replace the whole electronic board, probably with an updated version of the code. In this method, the tech support saves time, without the need to identify if the fault is in the software or hardware.

Recalling the example, in the maintenance of the washing machine, the worst case would be to manually test the pump, and if the pump is ok, the next step would be to replace the whole electronic board, without knowing if the fault is in the electronic board or software.

It is important to mention that there are tools which can verify the correctness of a software code *a priori* (Clarke, Kroening, & Lerda, 2004), however, there are faults which may still happen in the software. For example, in the manufacturing line of an embedded system, there is an operator which chooses a compiled archive which will be installed in the electronic board. Also, a determined product may have two kinds of electronic boards (e.g. a board with ethernet connection or other board with wifi connection). Usually, these boards have similar serial numbers, and it is possible that the operator download a wrong version of the code to the microcontroller. In this case, the consumer will receive a product which may work normally in most situations. Also, if this product is sent to the tech support, this problem may not be identified. Fault diagnosis techniques are most adequate to overcome this problem.

The multilayer diagnostic system (which is a part of the architecture presented in this work) takes advantage of the characteristics of the decentralized diagnosis structure (Zaytoon & Lafortune, 2013). In fact, the multilayer diagnostic system is a particular case of the decentralized diagnosis, without a coordinator. There is no need of a coordinator due to the independence between faults of different devices and layers in the embedded systems, which allows each subsystem and its respective diagnosers to be modeled independently. Furthermore, there is no exponential explosion in the state space size in the automata models, as the subsystems models are not composed, and therefore the method is suitable for large

embedded systems.

There are fault diagnosis approaches for the software (Yang, Bian, Li, Tan, & Tang, 2018), electronic hardware (Yan et al., 2018) or actuators (Lu et al., 2017) of embedded systems, however, to the best of our knowledge, we have not found a diagnosis architecture which considers the three layers of embedded systems.

In the work of Yang et al. (2018), they combine the artificial intelligence method of case-based reasoning with a Bayesian network to achieve the diagnosis of an embedded software system. They claim it can be extended to a multi-level diagnosis, which could be applied to the electronic board and actuators. On the other side, this technique is very complex, as it relates knowledge of artificial intelligence and statistics (Bayesian network).

The work of Yan et al. (2018) provides a technique of fault diagnosis in electronic boards, for faults caused by the effects of high power electromagnetic effects in the components of an electronic board. This technique is very specific and does not cover most applications of embedded systems.

The work of Lu et al. (2017) present a specific technique of online diagnosis of motor bearing. In other words, they develop a mathematical model which will only work on motors. This is the case for most fault diagnosis techniques of actuators, they are designed for a specific kind of actuator, based on their signals.

The main novelty of this work is the multilayer architecture for fault diagnosis which formally ensures that faults will be detected, isolated and identified according to each layer of the embedded system. We demonstrate the design of the diagnosis system applied to a home appliance refrigerator.

2. NOTIONS AND PRELIMINARIES

In this section we present the basic mathematical formulation, definitions and notation used in the article. For the reader which is not familiarized with the common mathematical notation of discrete event systems, it is highly recommended to read the work of Cassandras and Lafortune (2008).

2.1. Languages and Automata

Let $G = (X, \Sigma, \delta, \Gamma, x_0)$ denote a finite automaton, where X is the state space, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ is a finite event set that is partitioned into an observable and an unobservable events set, $\delta : X \times \Sigma \to X$ is a partial state transition function, $\Gamma : X \to 2^{\Sigma}$ is the function of active events and x_0 the initial state. Σ^* denotes the set of all possible sequences of finite length formed with events of Σ, including the empty trace ε. The prefix-closure of s is denoted by $\overline{s}$. The length of string s is denoted as $||s||$. $L(G)$ represents the language of G, where $L(G) \subset \Sigma^*$. σ_f represents a fault event and Σ_f represents a set of fault events. The set of all strings of L that ends with the event σ_f is denoted by $\Psi(\Sigma_f) = \{s\sigma_f \in L : \sigma_f \in \Sigma_f\}$. And, $L/s = \{t \in \Sigma^* : st \in L\}$ is the post-language of L after a string s.

The observer represents the observable behavior of G and is defined as $Obs(G, \Sigma_o) = (X_o, \Sigma_o, \delta_o, \Gamma_o, x_{0o})$, where $X_o \in 2^X$ (Cassandras & Lafortune, 2008).

The projection $P_o : \Sigma^* \to \Sigma_o^*$ is defined as $P_o(\varepsilon) := \varepsilon$; $P_o(\sigma) := \sigma$, if $\sigma \in \Sigma_o$, or $P_o(\sigma) := \varepsilon$, if $\sigma \in \Sigma_{uo}$; and $P_o(s\sigma) := P_o(s)P_o(\sigma)$, for $s \in \Sigma^*$, and $\sigma \in \Sigma$. The inverse projection is defined by $P_o^{-1}(t) := \{s \in \Sigma^* : P_o(s) = t\}$. The projection and inverse projection can be extended to a language L by applying its respective rules to all strings in L, respectively denoted by $P_o(L)$ and $P_o^{-1}(L_o)$.

2.2. Fault Diagnosis of Discrete Events Systems

The diagnosability concept refers to the capability to detect any fault in a system within a finite delay, based only on the occurrence of observable events recorded by the sensors. In terms of DES modeled by automata, a language generated by an automaton is diagnosable in relation to a set of observable events Σ_o and a set of fault events $\Sigma_f \subseteq \Sigma_{uo}$, if every fault event occurrence can be detected after the observation of a finite number of events occurrences.

Since the early studies related to fault diagnosis in DES, the following assumptions are made (Sampath, Sengupta, Lafortune, Sinnamohideen, & Teneketzis, 1995):

(A1) The language L generated by G is live. This means that there is a transition defined at each state x in X, formally we say $\Gamma(x) \neq \emptyset$ for all $x \in X$;

(A2) Automaton G has no cycle of unobservable events, i.e. $\forall ust \in L, s \in \Sigma_{uo}^*, \exists n_0 \in \mathbb{N}$ such that $||s|| \leq n_0$.

The assumption A1 is made considering that the system is always operating and the assumption A2 is necessary to prevent that the occurrence of the fault event ceases to be detected in case of the system staying stuck in a cycle of states formed only by unobservable events.

Considering $\sigma \in \Sigma$ and $s \in \Sigma^*$, we use the notation $\sigma \in s$ to denote that σ is an event of the string s. With a slight abuse of notation, we write that $\Sigma_f \in s$ to denote the fact that $\sigma_f \in s$ for some $\sigma_f \in \Sigma_f$ or formally $\overline{s} \cap \Psi(\Sigma_f) \neq \emptyset$. Finally, we can formally present the definition of diagnosability of a language (Sampath et al., 1995):

Definition 1 *Let L be a live and prefix-closed language. Then L is diagnosable with respect to the projection P_o and $\Sigma_f = \{\sigma_f\}$ if the following condition holds true:*

$$(\exists n \in \mathbb{N})(\forall s \in \Psi(\Sigma_f))(\forall t \in L/s)(||t|| \geq n \Rightarrow D), \quad (1)$$

where the diagnosis condition D is expressed by:

$$(\forall \omega \in P_o^{-1}(P_o(st)))(\Sigma_f \in \omega). \quad (2)$$

The definition presented above has the following meaning: Let s be an arbitrary string generated by the system that ends with a fault event belonging to the set Σ_f, and t is any sufficiently long continuation of the string s. Then the diagnosability condition D requires that all strings in L that generates the same record of observable events, such as the string st should contain in it a fault event of the set Σ_f. This implies that in all continuations t of s the occurrence of the fault can be detected within a finite delay.

One way to verify whether the occurrence of a fault (unobservable event) in a DES can be detected or not is by building a device called diagnoser. The diagnoser for G, is a deterministic automaton $G_d = (X_d, \Sigma_o, \delta_d, \Gamma_d, x_{0d})$ and can be obtained from $Obs(G||A_l)$, where A_l is a two-state label automaton as shown in Fig. 1.

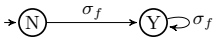


Figure 1. Label automaton A_l.

Note that the composition $G||Al$ generates states composed by $x \in X$ of G and $l \in X_{Al}$ of Al, in which $X_{Al} = \{\mathrm{Y}, \mathrm{N}\}$. For simplicity, the composed states are denoted as xl, where $x\mathrm{Y}$ indicate that the fault σ_f has occurred and $x\mathrm{N}$ indicate it has not occurred. When $G_d = Obs(G||A_l)$ is computed, the fault (unobservable) events are removed from the language. This procedure merge all states $q, p \in X_{G||Al}$ such that $\delta(q, \sigma_f) = p$ into $x_d = q, p$. Thus a state $x_d \in X_d$ is represented by $x_d = \{x_i l_1, x_2 l_2, \cdots, x_n l_n\}$ where $x_i \in X$ and $l_i \in X_{Al}$ for $i = 1, 2, \cdots, n$. A state is called certain (of fault) if $l = \mathrm{Y}$; and normal (i.e. non-faulty) if $l = \mathrm{N}$. If $\exists xl, y\tilde{l} \in x_d$, x not necessarily distinct of y, such that $l = \mathrm{Y}$ and $\tilde{l} = \mathrm{N}$, then x_d is an uncertain state of G_d.

The computational complexity of obtaining the diagnosers is, in the worst case, exponential in the cardinality of the state space of the system model. However, according to Paoli, Sartini, and Lafortune (2011), "the experience with applications of the diagnoser approach has shown that due to the structure of real systems, their diagnosers usually have a state space whose cardinality is of the same order as that of the original system". For more details about this topic, see Yin and Lafortune (2017).

Let $L(G, x) = \{s \in \Sigma^* : \delta(x, s) \in X\}$, i.e., the set of all strings that leads the automaton G from state $x \in X$ to any other state. Then a set of states $\{x_1, x_2, \ldots, x_n\} \subset X$ form a cycle in G if $\exists s$ such that $s = \sigma_1 \sigma_2 \ldots \sigma_n \in L(G, x_1)$ where $\delta(x_l, \sigma_l) = x_{l+1}, l = 1, \ldots, n-1$ and $\delta(x_n, \sigma_n) = x_1$.

A set of uncertain states $\{x_{d_1}, x_{d_2}, \ldots, x_{d_p}\} \subset X_d$ forms an indeterminate cycle if the following conditions hold true (Carvalho, Basilio, & Moreira, 2010):

(C1) $x_{d_1}, x_{d_2}, \ldots, x_{d_p}$ form a cycle in G_d, that is, there is $\sigma_l \in \Sigma_o$, $l = 1, 2, \ldots, p$, such that $\delta_d(x_{d_l}, \sigma_l) = x_{d_{l+1}}$, $l = 1, 2, \ldots, p-1$ and $\delta_d(x_{d_p}, \sigma_p) = x_{d_1}$;

(C2) $\exists (x_l{}^{k_l}, Y), (\tilde{x}_l^{r_l}, N) \in x_{dl}$, for $x_l^{k_l}$ not necessarily distinct from $\tilde{x}_l^{r_l}, l = 1, 2, \ldots, p, k_l = 1, 2, \ldots, m_l$, and $r_l = 1, 2, \ldots, \tilde{m}_l$ in such a way that the sequence of states $\{x_l^{k_l}\}, l = 1, 2, \ldots, p, k_l = 1, 2, \ldots, m_l$ and $\{\tilde{x}_l^{r_l}\}, l = 1, 2, \ldots, p, r_l = 1, 2, \ldots, \tilde{m}_l$, form cycles in G.

In words, an indeterminate cycle in a diagnoser is a cycle composed exclusively of uncertain states, which corresponds to the presence of two cycling traces in the system with the same observable projection, such that σ_f occurs in one trace but not in the other trace (Zaytoon & Lafortune, 2013).

Finally, the condition for diagnosability of G is determined in Theorem 1.

Theorem 1 *(Sampath et al., 1995) A language* L *generated by an automaton* G *is diagnosable with respect to its projection P_o and $\Sigma_f = \{\sigma_f\}$ if, and only if, its diagnoser G_d has no indeterminate cycles.*

2.3. Examples

In this section we show trivial examples to illustrate two different cases regarding the diagnosability of DES.

Consider the models G_1 and G_2 presented in Fig. 2a and Fig. 2b, respectively. For both models $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{f\}$.

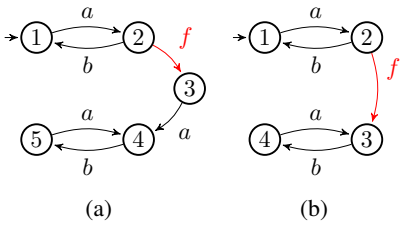


Figure 2. Automata models: (a) G_1 and (b) G_2.

For G_1, the normal behavior is contained in the states 1 and 2. The fault is diagnosable in this case, because after the occurrence of the fault, the sequence of events aa will be observed.

For G_2, the normal behavior is contained in the states 1 and 2. The fault is **not** diagnosable in this case, because after the occurrence of the fault, there is no observable event (nor string) which differs from the normal behavior. In other words, the observable strings in the faulty states shadows the normal behavior, making it impossible to distinguish between normal and faulty states.

In the following we show the diagnosers for G_1 and G_2, which can be used to formally verify the diagnosability of such systems. In Fig. 3a is displayed the diagnoser for G_1. There is no indeterminate cycle in G_{d1} and, therefore, the fault f in G_1 is diagnosable. In Fig. 3b is displayed the diagnoser for G_2. There is an indeterminate cycle in G_{d2} and, therefore, the fault f in G_2 is **not** diagnosable.

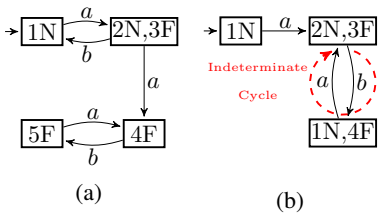


Figure 3. Diagnosers: (a) G_{d1} and (b) G_{d2}; for the respective models G_1 and G_2.

3. ARCHITECTURE FOR FAULT DIAGNOSIS IN EMBEDDED SYSTEMS

To allow the multilayer diagnostic system to be embedded in a device, it is necessary first to create a structure that allows the diagnosers access all data that is required to perform the diagnosis. In order to meet these conditions, we present a diagnostic architecture that considers the multilayer diagnostic system, as shown in Fig. 4.

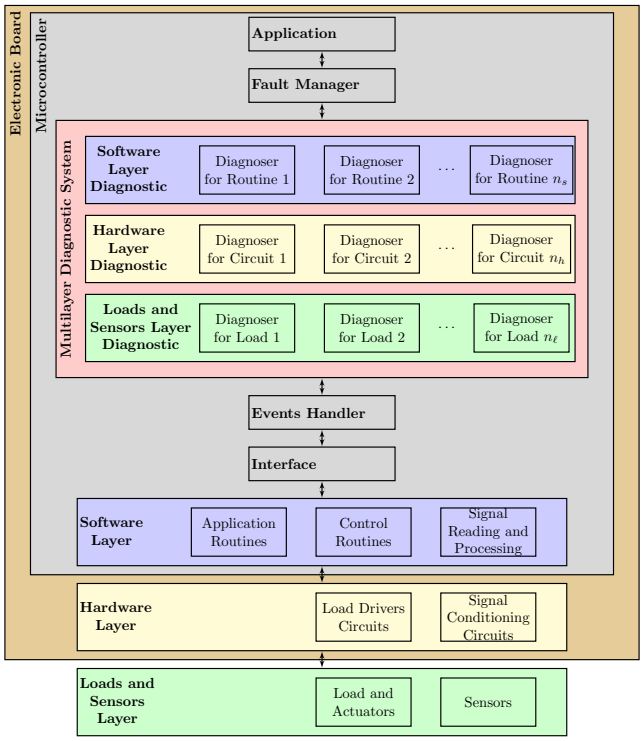


Figure 4. Multilayer architecture for the diagnosis of embedded systems.

In this architecture, the embedded system is divided in three main layers: the *Loads and Sensors*, *Hardware* and *Software* layers. In these layers are the purpose, or the final task of the embedded system.

The *Interface* block adequately manage the user requests taking into consideration the information of the actuators and sensors.

The *Events Handler* combines information of the layers from below (Fig. 4) in order to generate the necessary events in the format expected by the diagnosers.

The *Multilayer Diagnostic System* (MDS) monitors the system with modular diagnosers, which receive the information from the *Events Handler*. When a fault occurs, it is detected in the MDS and informed to the *Fault Manager* (FM) block.

The FM manages the information of the faults reported by each diagnoser in the MDS, and then communicates it to the *Application* block. Among the key attributes of the FM include: the ability to generate fault identification codes; organize the faults in a list according to a pre-established priority; generate a log with the situation of the system when a fault is reported by any diagnoser.

Finally, the *Application* block is responsible for providing an interaction with an external user. The user can see (e.g. in a display) the code of the fault; also, the use can reset the system after a maintenance is performed.

3.1. Multilayer Diagnostic System

We propose the *Multilayer Diagnostic System*, depicted in Fig. 4, whose aim is to isolate the source of the fault with respect to the layers of embedded systems. The multilayer diagnostic system will be called a deterministic multilayer diagnostic system if and only if the conditions imposed by Definition 2 are true.

Definition 2 *(Deterministic Multilayer Diagnostic System) Let* $\Sigma_f = \{\Sigma_{f_{sw}}, \Sigma_{f_{hw}}, \Sigma_{f_{\ell s}}\}$ *be the system fault set, where:* $\Sigma_{f_{sw}} = \{\sigma_{sw_i}\}$, *with* $i \in I = 1, ..., n_s$; $\Sigma_{f_{hw}} = \{\sigma_{hw_j}\}$, *with* $j \in I = 1, ..., n_h$; *and* $\Sigma_{f_{\ell s}} = \{\sigma_{\ell s_k}\}$, *with* $k \in I = 1, ..., n_\ell$ *denote respectively the fault set on the software, hardware and loads and sensors layers, and* n_s, n_h *and* n_ℓ *denote the number of faults on these sets. Also let* $G_{d_{sw_i}}$, $G_{d_{hw_j}}$ *and* $G_{d_{\ell s_k}}$ *be the diagnosers for the faults of the type* σ_{sw_i}, σ_{hw_j} *and* $\sigma_{\ell s_k}$, *respectively. The diagnostic system is a Deterministic Multilayer Diagnostic System if after the occurrence of a particular fault belonging to* Σ_f, *only its respective diagnoser:* $G_{d_{sw_i}}$, $G_{d_{hw_j}}$ *or* $G_{d_{\ell s_k}}$, *recognize it.*

To illustrate the Definition 2, let's recall the washing machine example. In the case of a fault in the pump, when it can not be turned on due to a broken coil winding, for example, the

diagnoser of this pump, allocated in the *Loads and Sensors Layer*, should detect this fault, and it is not allowed, for example, for the diagnoser of the pump drive circuit (*Hardware Layer Diagnostic*), nor, for the diagnoser of the pump control (*Software Layer Diagnostic*) to detect this fault.

4. APPLICATION: FROST FREE REFRIGERATOR

The proposed method is applied to a frost free refrigerator, manufactured by Whirlpool Corporation. The models were calculated in the software IDES (Integrated Discrete Event Systems) (Rudie et al., 2020). The diagnosers were implemented in software using C language, embedded in the same microcontroller where the main software of this refrigerator is located. In the following, we demonstrate the design for some of the diagnosers for this application according to the layers proposed in Section 3.

Diagnosers were designed for three routines in the *Software Layer*: thermostatic routine, compressor control routine and defrost routine. Considering the *Hardware Layer*, the diagnostic was designed for two circuits: compressor relay and defrost heater relay. For the *Loads and Sensors Layer*, diagnosers were designed for: the compressor and the defrost heater. In the following, the design process for one diagnoser of each layer is presented in order to illustrate specific techniques, such as sensor mapping. The fault events for each layer are listed in Table 1.

Table 1. Fault events modeled for the refrigerator application

Symbol	Event	Description
σ_{sw_1}	f_{TC}	Thermostatic control routine fault
σ_{sw_2}	f_{CC}	Compressor control routine fault
σ_{sw_3}	f_{DC}	Defrost control routine fault
σ_{hw_1}	f_{CRO}	Compressor relay stuck open
σ_{hw_2}	f_{CRC}	Compressor relay stuck closed
σ_{hw_3}	f_{DRO}	Defrost heater relay stuck open
σ_{hw_4}	f_{DRC}	Defrost heater relay stuck closed
$\sigma_{\ell s_1}$	f_{CP}	Fault in the compressor
$\sigma_{\ell s_2}$	f_{DH}	Fault in the defrost heater resistor

The thermostatic routine controls when the compressor must be turned *on* or *off* in order to keep the refrigerator temperature within the selected range. To perform this control, this routine reads the temperature value provided by the sensor and compares it with the reference values T_{on} and T_{off}. The automaton that represents the model of thermostatic routine G_{TC} is shown in Fig. 5a.

Next, the diagnoser Gd_{TC} is obtained by calculating $Gd_{TC} = Obs(G_{TC}||Al_{TC})$, shown in Fig. 5b. The fault event for the label automaton Al_{TC} is f_{TC}. The state 4Y represents the state where the diagnoser is certain of the fault occurrence. Note that there is a cycle of uncertain states {1N,4Y}, {2N,4Y} and {3N,4Y} $\in X_{Gd_{TC}}$, however it is not an indeterminate cycle, fulfilling the Conditions (C1) and (C2) of diagnosability. To exemplify this fulfillment, consider the automata G_{TC} and Gd_{TC} of Fig. 5a and 5b, respectively. The normal behavior of the G_{TC} is in states SATISF, COOLING or DEF. On the other side, if the system is deviated to a faulty behavior, the Gd_{TC} current state will be for sure 4Y. In other words, any deviation of the non-faulty behavior will lead to state 4Y of Gd_{TC}.

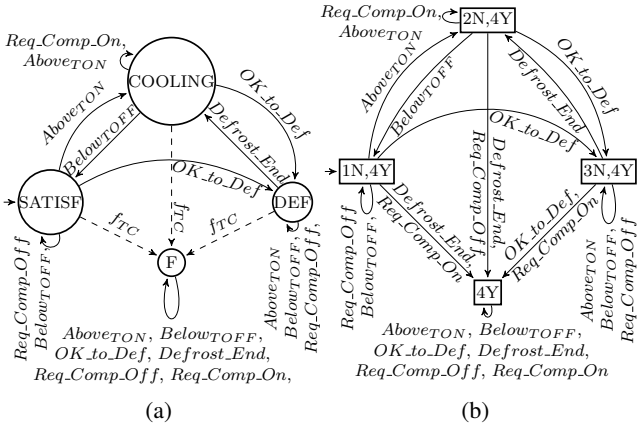


Figure 5. Automata models for (a) thermostatic routine G_{TC}; (b) thermostatic routine diagnoser Gd_{TC}.

The circuit board of this refrigerator has a feedback circuit, which reads the output signal of each relay. This feedback can be used to indicate the status of the relay, open or closed. This feedback signals were considered on the diagnosers design.

Since both relays have the same model, for simplicity a generic model and diagnoser will be used for both relays, where ℓ (load) must be understood as *Compressor* ($\ell = CP$) or *Defrost Heater* ($\ell = DH$) for each case. Two types of faults were considered for the relays: $f_{\ell RO}$ (Stuck Open) and $f_{\ell RC}$ (Stuck Closed). Also a sensor mapping (Sampath, Sengupta, Lafortune, Sinnamohideen, & Teneketzis, 1996) is considered in the *Events Handler* block, in which virtual events are generated depending on the state of the sensors, as shown in Table 2. The automaton that represents the relays models with sensor mapping is shown in Fig. 6a.

For the relays diagnosers the computation is given by $Gd_{\ell R} = Obs(G_{\ell R}||Al_{\ell RC}||Al_{\ell RO})$ and shown in Fig. 6b, where $Al_{\ell RC}$ with $\sigma_f = f_{\ell RC}$; and $Al_{\ell RO}$ with $\sigma_f = f_{\ell RO}$.

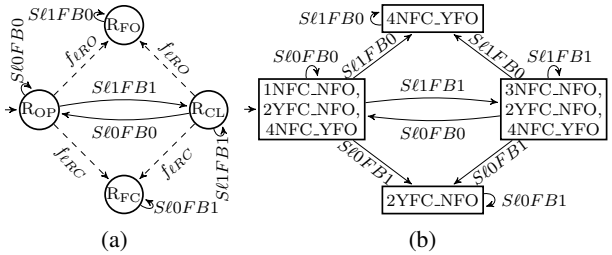


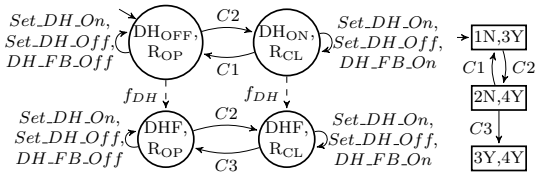
Figure 6. Automata models for (a) ℓ relay $G_{\ell R}$; (b) ℓ relay diagnoser $Gd_{\ell R}$.

Table 2. Sensors Mapping in the Events Handler Block

	State	Mapping	Event
ℓ Relay	$\mathrm{R_{OP}}$	$[Set\_\ell\_Off, \ell\_FB\_Off]$	$S\ell 0FB0$
	$\mathrm{R_{CL}}$	$[Set\_\ell\_On, \ell\_FB\_On]$	$S\ell 1FB1$
	$\mathrm{R_{OF}}$	$[Set\_\ell\_On, \ell\_FB\_Off]$	$S\ell 1FB0$
	$\mathrm{R_{CF}}$	$[Set\_\ell\_Off, \ell\_FB\_On]$	$S\ell 0FB1$
Defrost Heater	$\mathrm{DH_{OFF}}, \mathrm{R_{OP}}$	DH_FB_Off AND $[Def\_End\_By\_Temp$ OR $Def\_End\_By\_Timeout]$	$C1$
	$\mathrm{DH_{ON}}$, $\mathrm{R_{CL}}$	DH_FB_On AND Def_Start	$C2$
	DHF, $\mathrm{R_{OP}}$	DH_FB_Off AND $Def\_End\_By\_Timeout$ AND $Def\_End\_Temp < T_{ref}$	$C3$
	DHF, $\mathrm{R_{CL}}$	DH_FB_On AND Def_Start	$C2$

In order to distinguish the heater fault and the heater relay fault, it is necessary to consider also the sensor mapping in the *Events Handler* block, as shown in Table 2. This mapping was obtained by creating a virtual sensor to generate the events of the defrosting routine and using the information from defrost sensor. In Fig. 7a the automaton G_{DH} that models defrost heater is depicted.

The label automaton Al_{DH} is obtained considering $\sigma_f = f_{DH}$. Thus the diagnoser Gd_{DH} shown in Fig. 7b is obtained by calculating $Gd_{DH} = Obs(G_{DH}||Al_{DH})$.



(a) (b)

Figure 7. Automata models for (a) defrost heater G_{DH}; (b) defrost heater diagnoser Gd_{DH}.

4.1. Results

In this Section we describe the methodology and results of the tests realized to validate the proposed architecture.

The automata models in the *Multilayer Diagnostic System*, as well as the sensor mapping in the *Events Handler* block, designed in the Section 4, were automatically translated to *C code* through the software tool *Doctor Who*, which was developed in the context of this work.

For the validation of the proposed method it was used the STVD (ST Visual Develop, from the ST Microelectronics manufacturer) which allows to online edit and monitor variables of the code running in the microcontroller. Along with the STVD, it was developed, in the context of this work, the

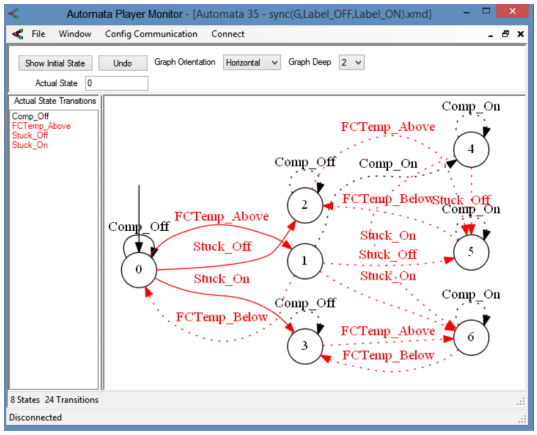


Figure 8. Screenshot of the *automata player* software tool, which can monitor events online, through a serial communication with the microcontroller.

software tool *Automata Player Monitor* which can monitor the events of the refrigerator through a serial communication and display it in the graphical representation of the automaton in an online environment, as shown in Fig. 8.

In order to verify the proposed method, the frost free refrigerator prototype was submitted to exhaustive tests in Whirlpool's laboratory. The tests were designed to force faulty situations (regarding all modeled faults) to verify the effectiveness of the multilayer diagnostic system. In total, 9 different kinds of faults were tested.

For the *Software Layer* faults, bugs were intentionally placed in the code.:

- Bug insertion in the thermostatic routine;
- Bug insertion in compressor control routine;
- Bug insertion in the defrost routine.

For the *Hardware Layer* faults, the input and output of the relays were short-circuited in the case of *relay stuck closed*. For the *relay stuck open* faults, the relays coils were disconnected from the circuit, keeping the relay open in this case:

- Compressor control relay locked closed;
- Compressor control relay locked open;
- Defrost resistor relay locked closed;
- Defrost resistor relay locked open.

For the *Loads and Sensors Layer* faults, the loads were disconnected from the circuit to simulate the broken filament (defrost heater) or damaged coil (compressor):

- Broken heating filament for the defrost resistance;
- Damaged internal starter coil of the compressor.

4.2. Test plan

In this section we precisely describe the test procedure, pointing out how the faults were tested, and how many times they were ran.

Test 1:

- Initial condition: Temperature below the control point;
- 5 door opening of 30s every hour.

Test 2:

- Initial condition: Temperature above the control point;
- 5 door openings every 30s every hour.

Test 3:

- Initial condition: Temperature above the control point;
- Door closed for 6 hours.

Test 4:

- Initial condition: Temperature below the control point;
- Door closed for 6 hours.

Test 5:

- Closed doors;
- 2 defrost cycles.

Test 6:

- Opening doors (random);
- 2 defrost cycles.

Test 7: Fast freezer mode activated (this forces the compressor to stay on for 6 hours straight and inhibits defrosting during this period).

Test 8: Holiday mode activated. This forces you to decrease the compressor running time and make adaptive defrosts (based on longer times).

Test 9: Forcing emergency defrost condition. The defrost is forced by time, even without temperature conditions.

Each test was repeated for each type fault:

$$9\ faults \times 9\ tests = 81. \tag{3}$$

Each test was repeated 3 times for each setpoint (Min, Med and Max), except for tests 7 and 8 which are not applicable to setpoints but to *holiday mode* and *fast freezer mode* only:

$$\begin{aligned} 9\ faults \times 7\ tests \times 3\ setpoints + \\ + 9\ faults \times 2\ tests = 207 \end{aligned}$$

The whole procedure was repeated combining each failure two by two (9 tests combined 2 by 2 = 36 combinations):

$$\begin{aligned} 36\ comb. \times 7\ tests \times 3\ setpoints + \\ + 36\ comb. \times 2\ tests = 828\ runs \end{aligned}$$

In total, 1035 runs where executed.

Results show that each fault was precisely diagnosed according to the forced faulty situations, correctly detecting, isolating and identifying the faults. Furthermore, we have not observed any false positives, nor false negatives diagnostics, and therefore there is no need of a confusion matrix.

5. CONCLUSION

The multilayer architecture is adequate for diagnosis of embedded systems. The main advantage of this method is to achieve an isolation of the fault, regarding the three layers of embedded systems: software, hardware and loads & sensors. The necessary conditions to define the diagnostic system as deterministic multilayer diagnostic system were presented. The proposed method was applied in a case study, in which exhaustive practical tests were made. In a laboratory test, faulty situations for all modeled faults were forced in the prototype. Results show that all modeled faults were precisely detected, isolated and identified with respect to its own layer and device. The results of this work led to an application of this method in home appliance refrigerators, which were produced in industrial scale.

Whirlpool (Joinville) have included this method in their refrigerators, with slightly differences to: adequate the method to their product operational system; to use software which they already own licenses; and, to adequate the method to the developers knowledge to facilitate the code reading and maintenance. The obtained advantages were relevant, and Whirlpool's (Joinville) refrigerator quality became a reference to other Whirlpool's plants.

ACKNOWLEDGMENT

This work was supported by Whirlpool Latin America and by Santa Catarina State University (UDESC). Second author acknowledges UNIEDU/FUMDES for PhD scholarship program of the Santa Catarina State. The third author is thankful for the partial financial support from Fundação de Amparo à Pesquisa e Inovação do Estado de SC, Brazil - FAPESC 2021TR930.

REFERENCES

Ammour, R., Leclercq, E., Sanlaville, E., & Lefebvre, D. (2017). State estimation of discrete event systems for RUL prediction issue. *International Journal of Production Research*, *55*(23), 7040 - 7057.

Bennouna, O., & Roux, J. (2013). Real time diagnosis & fault detection for the reliability improvement of the embedded systems. *Journal of Signal Processing Systems*, *73*(2), 153 - 160.

Carvalho, L. K., Basilio, J. C., & Moreira, M. V. (2010). Robust diagnosability of discrete event systems subject

to intermittent sensor failures. *IFAC Proceedings Volumes*, *43*(12), 84 - 89.

Cassandras, C. G., & Lafortune, S. (2008). *Introduction to discrete event systems* (2nd ed.). Kluwer Academic Publishers.

Clarke, E., Kroening, D., & Lerda, F. (2004). A tool for checking ANSI-C programs. In K. Jensen & A. Podelski (Eds.), *Tools and algorithms for the construction and analysis of systems* (pp. 168–176). Berlin, Heidelberg: Springer Berlin Heidelberg.

Fritz, R., & Zhang, P. (2018). Overview of fault-tolerant control methods for discrete event systems. *IFAC-PapersOnLine*, *51*(24), 88-95.

Gandhi, P., Turk, D. N., & Dahiya, D. R. (2020). Health monitoring of induction motors through embedded systems-simulation of broker rotor bar fault and abnormal gear teeth fault. *Microprocessors and Microsystems*, *76*, 103077.

Ge, N., Nakajima, S., & Pantel, M. (2015). Online diagnosis of accidental faults for real-time embedded systems using a hidden Markov model. *Simulation*, *91*(10), 851 - 868.

Goebel, K., & Rajamani, R. (2021). Policy, regulations and standards in prognostics and health management. *International Journal of Prognostics and Health Management*, *12*(1).

Guo, Y., Wang, J., Chen, H., Li, G., Huang, R., Yuan, Y., ... Sun, S. (2019). An expert rule-based fault diagnosis strategy for variable refrigerant flow air conditioning systems. *Applied Thermal Engineering*, *149*, 1223 - 1235.

Lu, S., He, Q., Yuan, T., & Kong, F. (2017). Online fault diagnosis of motor bearing via stochastic-resonance-based adaptive filter in an embedded system. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *47*(7), 1111 - 1122.

Lu, S., He, Q., & Zhao, J. (2018). Bearing fault diagnosis of a permanent magnet synchronous motor via a fast and online order analysis method in an embedded system. *Mechanical Systems and Signal Processing*, *113*, 36 - 49.

Lu, S., Qian, G., He, Q., Liu, F., Liu, Y., & Wang, Q. (2020). In situ motor fault diagnosis using enhanced convolutional neural network in an embedded system. *IEEE Sensors Journal*, *20*(15), 8287 - 8296.

Moreira, B. G., & Leal, A. B. (2020). A proposal for an active diagnoser for safe fault-tolerant control of discrete event systems. *IFAC-PapersOnLine*, *53*(4), 282-287.

Naha, A., Thammayyabbabu, K. R., Samanta, A. K., Routray, A., & Deb, A. K. (2017). Mobile application to detect induction motor faults. *IEEE Embedded Systems Letters*, *9*(4), 117-120.

Ning, S., Han, Z., Wu, X., & Wang, Z. (2018). Gear crack fault diagnosis based on embedded sensors. *Zhendong yu Chongji/Journal of Vibration and Shock*, *37*(11), 42 - 47.

Paoli, A., Sartini, M., & Lafortune, S. (2011). Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, *47*(4), 639-649.

Pons, R., Subias, A., & Trave-Massuyes, L. (2015). Iterative hybrid causal model based diagnosis: Application to automotive embedded functions. *Engineering Applications of Artificial Intelligence*, *37*, 319 - 335.

Ranade, A., Provan, G., El-Din Mady, A., & O'Sullivan, D. (2020). A computationally efficient method for fault diagnosis of fan-coil unit terminals in building heating ventilation and air conditioning systems. *Journal of Building Engineering*, *27*, 100955.

Rudie, K., Bretzke, H., Dragert, C., Edlund, K., Grigorov, L., McAloney, C., ... Wood, M. (2020). *Integrated discrete-event systems software.* GitHub. Retrieved from `https://github.com/krudie/IDES`

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, *40*(9), 1555-1575.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. C. (1996). Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, *4*(2), 105-124.

Takai, S. (2021). A general framework for diagnosis of discrete event systems subject to sensor failures. *Automatica*, *129*, 109669.

Vignolles, A., Chanthery, E., & Ribot, P. (2020). An overview on diagnosability and prognosability for system monitoring. In *Proceedings of the european conference of the PHM society 2020* (Vol. 5, p. 11).

Watanabe, A. T. Y., Leal, A. B., Cury, J. E. R., & de Queiroz, M. H. (2021). Combining online diagnosis and prognosis for safe controllability. *IEEE Transactions on Automatic Control*.

Watanabe, A. T. Y., Sebem, R., Leal, A. B., & Hounsell, M. d. S. (2021). Fault prognosis of discrete event systems: An overview. *Annual Reviews in Control*, *51*, 100-110.

Yan, J., Wang, J., Tang, C., Liu, X., Yang, M., Hao, W., ... Zeng, H. (2018). Performance investigation of VCSEL-based voltage probe and its applications to HPEM effects diagnosis of embedded systems. *IEEE Transactions on Electromagnetic Compatibility*, *60*(6), 1923 - 1931.

Yang, S., Bian, C., Li, X., Tan, L., & Tang, D. (2018). Optimized fault diagnosis based on FMEA-style CBR and BN for embedded software system. *International Journal of Advanced Manufacturing Technology*, *94*(9), 3441 - 3453.

Yin, X., & Lafortune, S. (2017). Verification complexity of a class of observational properties for modular discrete

events systems. *Automatica*, *83*, 199-205.

Zaytoon, J., & Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, *37*(2), 308 - 320.