HAL
open science

Functional-architectural diagnosability analysis of embedded architecture

Manel Khlif, M. Shawky

To cite this version:

Manel Khlif, M. Shawky. Functional-architectural diagnosability analysis of embedded architecture. Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on, Oct 2011, Washington, DC, United States. pp.469 - 476, 10.1109/ITSC.2011.6082819 . hal-00801610

HAL Id: hal-00801610

https://hal.science/hal-00801610v1

Submitted on 18 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Functional-Architectural Diagnosability Analysis of Embedded Architecture

Manel KHLIF, M. SHAWKY
Heudiasyc-UMR CNRS 6599
Université de Technologie de Compiègne
Compiègne, France
{manel.khlif; shawky}@hds.utc.fr

***Abstract*—Diagnosability analysis of functions offers now a serious complement to knowledge-based methods of diagnosis, such as FMEA (Failure Mode and Effects Analysis) and fault tree analysis. State of the art of diagnosability analysis focus on what we call "functional diagnosability", where the hardware architecture of the system and its constraints are not directly considered.**

This paper contributes to the analysis of the functions-architecture interaction impact on the diagnosability of an embedded system, especially automotive systems. The approach we developed can be integrated into the design cycle. It has two important phases; first, the diagnosability analysis of discrete event systems, then the verification of a property set that we have defined and called the "diagnosability functional-architectural properties". Properties verification is done in two stages: check the description of the architecture, described in AADL, and check the functions-architecture interaction, modeled in SystemC-Simulink. The validation process is applied on a real automotive experimental embedded platform based on several Electronic Control Units.

Finally, we have developed through this paper a novel methodology for the analysis of diagnosability that takes into account the constraints of the hardware architecture of the system.

I. INTRODUCTION

IN complex automated systems failures are increasingly difficult to predict, understand and repair. The need for methods and tools for the supervision of these systems has initiated many research projects. Thus, the implementation of these systems was naturally modular to control the complexity and risk, hoping that the errors do not affect all parts of the modules. However, this modularity, or distribution of functions, has contributed to increase the vulnerability of this type of system.

Research works on dependability helped to develop verification techniques to control hazards. Most of these techniques of diagnosis are knowledge-based (rules-based systems, fault dictionary, etc.)[1]. At the same time, other research works undertaken to improve the reliability of these systems by reviewing the design methodologies. Diagnosis methods have evolved to model-based approaches [2] that deal better with distribution. To improve the fault tolerance in embedded systems and their ability to self-diagnose, the field of "diagnosability" analysis has emerged. Nowadays, the system designer must ensure that the system is diagnosable, i.e. must ensure that the errors that may appear are identifiable. Diagnosability is considered as a requirement to be verified during the design, as important as the properties related to dependability (safety, reliability, etc.) to provide a more reliable system, with predictable maintenance costs.

Methods of diagnosability analysis focus on what can be called "functional diagnosability", where the hardware architecture was not directly considered [3]. In fact, the classical diagnosability is a property defined on the paths representing a system. It specifies that whenever a fault may occur, it exists a finite set of observations that allows us to decide whether this fault did happen or not [4]. However, the need to analyze the diagnosability of hardware-software architecture has been discussed mainly by transportation industry. Indeed, the automotive industry reported the problem of on-board diagnosis implementation because of the large distribution of functions (on computing units), the large interaction of functions with sensors and actuators (through communication bus) and real time constraints. Model-based diagnosis is the answer to this problem and led to the need of diagnosability analysis.

Hence, we are interested in a specific point: Do hardware architecture and even the couple software-hardware architecture influence system diagnosability?

In this paper, we present a method of functional-architectural diagnosability analysis. We discuss our method that makes confrontation between the architecture description and the classical model of diagnosability (functional model). We define a property set that takes into consideration the architecture distribution and the time constraint, to analyze the functional-architectural diagnosability.

II. FUNCTIONAL DIAGNOSABILITY

The diagnosability research area is recent. Several communities have developed different approaches for model-based diagnosis. Indeed, for the diagnosability analysis of embedded electronic architectures, the most commonly used approaches are using "discrete-event based" models, not considering directly the hardware architecture board at the system description.

In order to perform a functional diagnosability analysis, a behavioral model of the system is needed. As indicated before,

multiple models can be envisioned: continuous-state-based, event-based, or hybrid. When dealing with the diagnosability of discrete event systems, the most widely accepted analysis method is the "Diagnoser" approach due to M. Sampath et al [4]. The input model is a finite deterministic state machine modeling the system behaviors (Figure 1).

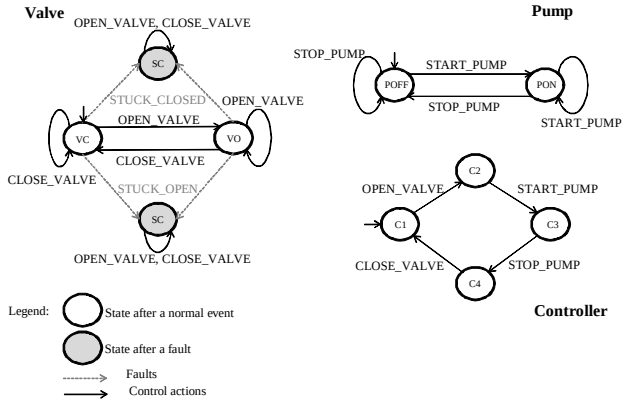

Figure 1. Behavioral components models of PVC (Pump, Valve, Controller) sub-system of HVAC (Heating, Ventilating, Air Conditioning) system [4]

The analysis is performed by augmenting that model with non-normal or fault states and checking if the augmented and transformed model is diagnosable.

III. Functional-Architectural Diagnosability Properties Definition

A. *Architecture in diagnosability*

While the system model being functionally diagnosable and the existence of a diagnoser are necessary conditions for the final embedded system to be diagnosable by itself, they are not sufficient. The interaction between the functions, the diagnoser and their mapping to the underlying architecture has to be considered. It is thus necessary to define checkable properties that the architecture has to demonstrate with respect to the functions and the diagnoser in order to be diagnosable.

To verify the functional-architectural diagnosability, we propose a set of properties to check used as a complement to the method of verification of diagnosability of discrete event systems cited in [4]. We focus on distributed embedded architectures based on computing units connected with one or more communication networks.

These properties are necessary for the different diagnosis structures (i.e. centralized, decentralized and distributed) and involve the notion of time for certain scenarios. The first step in the properties verification, is determining the diagnosis structure (centralized, decentralized or distributed) to identify all the necessary properties. Thus, an architecture is called diagnosable if and only if all the properties concerning its structure are verified.

B. *Properties defintion*

In order to define properties, we first define the system using a set of variables and functions representing the different components and characteristics.

Variables definition:
A : All computing units (or component) containing one or more functions to analyze
B : All components sources of state variables
C : All components that are computing units
D_principal : The computing unit (or component) diagnoser (Case of centralized and decentralized diagnosis)
D : All computing units (or components) diagnosers
F : All functions
F_Diag : All diagnosis functions
M : All messages sent through the communication bus
O : The computing unit (or component) containing the observer process
V : The set of state variables issued from sensors or other components of the system

Types definition :
void : Undefined return type, not null

Functions definition:
F_Data : $(x,y)\rightarrow\{0,1\}\backslash\ (x,y) \in C^2$ returns 1 if x and y are connected by a data connection, otherwise returns 0
Size_F : $x\rightarrow y\backslash\ x \in F_Diag,\ y \in \mathbb{R}$ returns the size of x
Size_Mem : $x\rightarrow y\backslash\ x \in C,\ y \in \mathbb{R}$ returns the memory size of x
ΔT : $x\rightarrow y\backslash\ x \in F_Diag,\ y \in \mathbb{R}$ function that returns the desired frequency for performing the function x
WCET : $(x,y)\rightarrow z\backslash\ x \in F_Diag,\ y \in C,\ z \in \mathbb{R}$ returns the maximum execution time that x can make on y platform, it is the WCET (Worst-Case Execution Time) of x on y[9]. We assume the existence of this function.
Conn_I/O : $(x,y)\rightarrow z\backslash\ x\in C,\ y \in B,\ z \in\{0,1\}$ returns 1 if x and y are connected
Implemented : $(x,y)\rightarrow z\backslash x\in F,\ y \in C,\ z \in \{0,1\}$ returns 1 if x is implemented on y
Source : $x\rightarrow y\backslash\ x\in M,\ y\in C$ returns the name (or the address) of the computing unit sender of the message x.
Destination : $x\rightarrow y\backslash\ x\in M,\ y\in C$ returns the name (or address) of the computing unit receiver of the message x
Content : $x\rightarrow y\backslash\ x\in M,\ y\in$ « void » returns the content of the message M.
Value : $x\rightarrow y\backslash\ x\in V,\ y\in$ « void » returns the value of the variable V.
Origin : $x\rightarrow y\backslash\ x\in \{E, V\},\ y\in B$ returns the component source of the event or the state variable

Properties must be checked in the following order:

Rule 1 : For every diagnosis structure, we have a properties set to verify :

- Centralized : properties 1.1, 2, 3, 4, 5 and 6
- Decentralized : properties 1.2, 3, 4, 5 and 6
- Distributed : properties 1.3, 2, 3, 4, 5 and 6

Property 1. Connectivity to diagnoser : Checking this property ensures that every hardware component executing a

function to diagnose is connected to the diagnoser component. That connection depends on the structure of the diagnoser:

- In a centralized structure, each hardware component executing a diagnosed function must have at least one data connection to the diagnoser component.
- In a decentralized structure, each secondary diagnoser component must have at least one data connection to the primary diagnoser component.
- In a distributed structure, each hardware component executing a diagnosed function must also be a diagnoser component.

➢ ***Property 1.1 :*** For any component (computing unit) containing a function to diagnose, we must have at least one "data" connection with the component (computing unit) diagnoser (1) .

$$\forall a \in A, \\ (F_Data\ (a, D_principal)) \vee (a==D_principal) \\ \rightarrow Connectivity_Diagnoser\ (a) = 1 \tag{1}$$

➢ ***Property 1.2 :*** For any component (Computing unit) of auxiliary diagnosis, we must have at least one "data" connection with the principal diagnose (2).

$$\forall d \in D, \\ ((Conn_I/O\ (d, D_principal) == 1) \\ \rightarrow Connectivity_Diagnoser\ (d) = 1 \tag{2}$$

➢ ***Property 1.3 :*** For any component (computing unit) containing a function to diagnose, it must be itself a component diagnose(3).

$$\forall a \in A, \forall d \in D, \\ (a==d) \\ \rightarrow Connectivity_Diagnoser\ (a) = 1 \tag{3}$$

Property 2. Executability : This property ensures that every diagnosis function can indeed be executed on the computing units it has been mapped to. It is twofold:

- The diagnoser memory footprint must not be higher than the memory available on the computing unit
- The WCET of the diagnoser must not be higher than its minimum desired execution period.

If any of the two sub-properties above reaches the limits of the computing unit, it has to be entirely devoted to the diagnoser function and cannot be shared (4).

$$\forall d \in D, \forall f \in F_Diag, \\ ((Implemented\ (f,d)==1) \wedge (Size_Mem\ (d) \geq Size_F\ (f)) \wedge (\Delta T\ (f) >= WCET\ (f,d))) \\ \rightarrow Executability\ (f, d) = 1 \tag{4}$$

Property 3. Reachability : In our approach, we see that the decision of the observability of an event, in the model (the finite automaton) representing the system, do not stop at the only information issued from the sensors, because the whole architecture hardware and software, in interaction with the sensors, can have an important role in observing system events.

Thus an "observer" process (Figure 2), part of the diagnoser must be implemented and must achieve (receive) the values of all state variables representing the information from sensors or other system components (Figure 3).

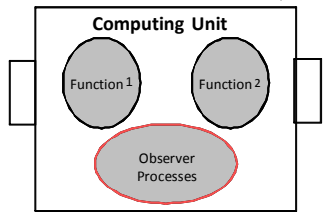

Figure 2. Observer process

The "observer" process must receive the values of state variables of all components of the system to analyze. A state variable is called reachable when the hardware component including the origin of this variable is physically reachable by the hardware component on which the observer is located (5).

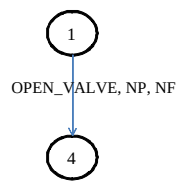

Figure 3. Event example
OPEN_VALVE: variable representing the event
NP, NF: state variables representing the information issued from the sensor

$$\forall v \in V, O \in C, \forall m \in M, \\ (Conn_I/O\ (Origin\ (v), O) \vee F_Data\ (Origin\ (v), O)) \wedge \\ (Source\ (m) = Origin\ (v)) \wedge (Destination\ (m) = O) \wedge (Content\ (m) = Value\ (v)) \\ \rightarrow Reachability\ (v) = 1 \tag{5}$$

Property 4. Accessibility : An observer has to be notified of all the events occurring within the diagnosed components. This property is similar to the reachability, but pertains to event variables instead of state variables. An event is said to be accessible when the hardware component implementing the observer can physically access the originating component for that event (6).

$$\forall e \in E, \forall c \in C, \forall m \in M, \\ (Conn_I/O\ (Origin\ (e), O) \vee F_Data\ (Origin\ (e), O)) \wedge (Source\ (m) = Origin\ (e)) \wedge (Destination\ (m) = O) \wedge (Content\ (m) = Value\ (e)) \\ \rightarrow Accessibility\ (e) = 1 \tag{6}$$

Property 5. Temporal availability : This property is verified when the available time on the computing unit intended for the diagnosis function is sufficient for the execution of a diagnoser (or a secondary diagnoser in a decentralized structure). To verify this property, all the time slices not used by the nominal operation of the system are considered and their total length compared to the diagnoser needs [5].

Property 6. Observability : All event variables from the automaton (Figure 3) have to be observable for the whole event to be itself observable. Observability is thus a compound property, deriving from the previous ones. It is verified if:

- the temporal availability of the system is sufficient to implement a diagnoser,
- its state variables are reachable,
- its events are accessible [5].

Rule 2: Every property of functional-architectural diagnosability is verified through system representation or modeling. Properties {1.1, 1.2, 1.3, 2, 3, and 4} are verifiable at the architecture description level, while the properties {5 and 6} are verifiable at the level of functions-architecture interaction:
•Verification of the architecture description: no time, made from the architecture description (or documentation).
•Verification of the functions-architecture interaction: takes into account the progress of the state of the hardware-software system through time.

IV. ARCHITECTURE MODEL FOR DIAGNOSABILITY

A. Architecture description and associated tools

Most of the architectural properties the diagnosability depends on are of a static nature. They are only affected by the structure of the hardware architecture and the location of the diagnoser within that architecture. Therefore, the models used to evaluate these properties must be able to capture the physical organisation of the architecture, including its computing units, communication buses and devices such as sensors and also their interaction, through description of data and event flows. Several description languages can be used for this purpose.

In this paper the focus is essentially on AADL (Architecture Analysis & Description Language) [6], but the methodology could be applied from models expressed using VHDL (Very high speed integrated circuit Hardware Description Language) [7] or EAST-ADL for instance, as both of them support hardware architecture description.

The case for AADL comes from the fact that it is getting a growing attention both in the avionics and automotive fields and that it is specifically tailored for hardware architecture description, ignoring its proposed behaviour annex.

B. Functions-architecture interaction modelling and associated tools

While architecture description is enough to cater for the first four properties (1 to 4), the temporal availability property can only be evaluated by considering the whole system, including the interaction of the functions with the hardware architecture.

The architecture description model discussed above has to be augmented to also include the behaviour of the functions and the use they make of the hardware resources as they are mapped onto the hardware architecture. The resulting model should be suitable for either formal analysis or simulation-based analysis. The latter approach is often preferred, as tools are readily available that can be used to generate simulation traces useful for further analysis.

One idea is to describe the functional behaviour using the same language or modelling framework used to describe the architecture. The advantage of this idea is that it leads to a complete, homogenous functional-architectural model of the system to be analysed. In the case of VHDL for instance, a lot of simulators are available that can be used to provide execution traces for analysis. However, the functional model is rarely expressed from the start in such languages. For instance, MATLAB/Simulink is widely used in the industry for such a task. In order to again reduce the need for model translation and adaptation, a heterogeneous approach is preferred.

As AADL is not really suitable for such a goal, because it cannot describe processes behavior, other languages can be considered. If we keep MATLAB/Simulink functional models in mind, a SystemC model can be used for co-simulation and trace generation as described for instance by J.F Boland [10] by making calls to the MATLAB environment from the SystemC model. While translating an AADL model to SystemC can be done by hand, there is ongoing work towards automatic translation tools, such as AADS [11].

V. ARCHITECTURAL-FUNCTIONAL DIAGNOSABILITY VERIFICATION

A. Automata and architecture description matching

The first step is to parse the AADL model to extract and identify all the architectural features necessary to the analysis, such as the embedded computing units, devices such as sensors, communication links, data flows and inputs/outputs connections representing state and event variables.

Using that information, each state and event variable found in the functional model (automata) can be matched to the corresponding data flow in the AADL model, including its origin, its destinations, the physical links supporting the data or message transmission, etc.

B. Properties verification

1) Property verification at the architecture description level

The architecture properties are evaluated using that matching mentioned previously. For example, in order to check the accessibility property for a given event variable, we have to check that a data flow exists for that variable, i.e. it is actually output by some component, that the data flow actually reaches the observer, i.e. there exists a connection between its source and the observer or its source and the observer are mapped to the same computing unit. The global accessibility

property is evaluated by repeating that process for each event variable, one at a time. Checking for other properties follow similar processes.

2) Property verification at the functions-architecture interaction level

We proceed to the analysis of properties that depend on the architecture and on the evolution of the system state in time. To do this we had a choice between developing our own analysis engine analysis or rely on existing engines. We have chosen to use simulation engines compatible with the formalism of models expression of the most practiced in industry (such as Matlab / Simulink, etc.). We therefore use different simulation engines to "unroll" the system behavior according to his model and according to simulation inputs representing critical scenarios. To develop relevant scenarios for simulation cycles, we are inspired by some of the techniques MC / DC (Modified Condition/Decision Coverage) for the selection of inputs values, and for the events sequence [14]. MC/DC is a testing process that meets a source code coverage criterion; it is used by the DO-178B standard [13]. Therefore, we focused our efforts on the interpretation of simulation results of a model. Indeed, when we simulate a model, we can produce a log file that records all changes, precisely dated values of the signals.

The main point of that approach is that it can be completely automated, making its integration into a tool chain possible. While hand checking is still possible, it would be very time-consuming and error-prone. Tools, such as our prototype COSITA (CO-Simulation Trace Analysis) [8], are still a work in progress, but are essential if diagnosability is to be considered as a useable metric in an architecture exploration process. We developed the tool suite COSITA where iterated co-simulations traces are analyzed to evaluate the interaction of functions with the architecture, according to selected critical scenarios. We compared results of co-simulation trace analysis with results that we obtained through a physical emulation automotive platform.

VI. CASE STUDY

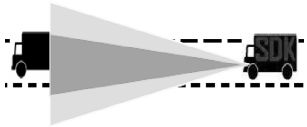

Figure 4. SDK (Smart Distance Keeping) function

We have tested our approach on the Smart Distance Keeping (SDK) function, given by a truck manufacturer, as a part of a national project. “SDK” is equivalent to the Adaptive Cruise Control (ACC) function, except that the distance/velocity regulation is based only on a fixed distance of 50 m (compliant to European regulations for heavy trucks) [12]. Thus, using embedded radar, the SDK sub-system maintains a safe headway time, i.e. the inter-vehicle distance is varying as a function of the velocity and is maintained at a minimum legal distance of 50 m (Figure 4). The final purpose of this case study, in the project, is to make a real time on-board-diagnosis of this function based on system model.

We have first verified the “functional” diagnosability of the SDK, through the approach of M. Sampath (using automata). Hence, we model the SDK function using a set of finite automata. A partitioning of the diagnosability analysis of the system must be made to identify all the components to be analyzed. Other components in interactions may be part of another diagnosability analysis, such as engine function, or braking function diagnosability analysis, etc....SDK includes the following components: radar, computing units, the vehicle velocity value and engine. Expected “Faults” events are:

- « Radar failure » : F1 leads to the state « Failure 1»
- « Velocity_measure failure »: F2 leads to the state « Failure 2» (Figure 5).

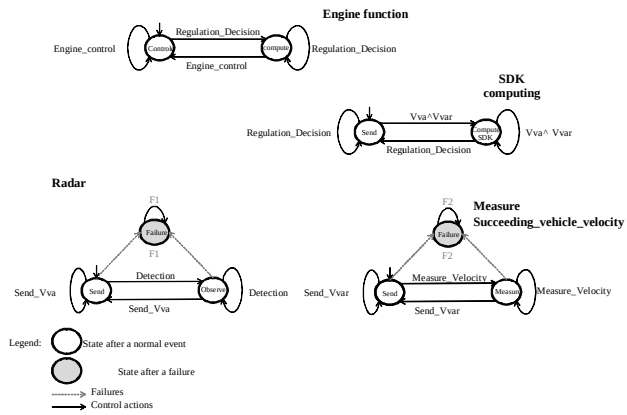

Figure 5. Behavior model of SDK components
Vvar : Succeeding vehicle velocity
Vva : Preceding vehicle velocity

The result of the analysis is positive: SDK is functionally diagnosable. After that, we described the electronic architecture of SDK (Figure 6) with AADL language.

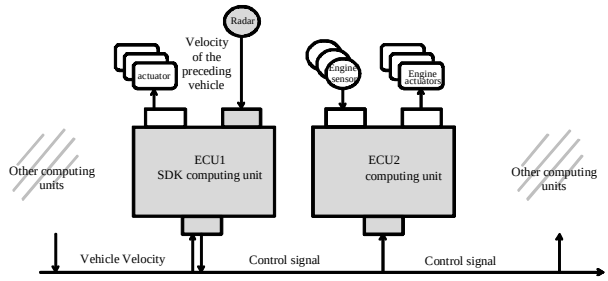

Figure 6. Different electronic components of SDK

We described also messages exchange between computing units in pseudo code:

- The two messages containing state variables values,

```
Message_1 :
Source : ECU1
Destination : ECU1
Content : Value(RV)
Message_2 :
Source : ECU1
Destination : ECU1
Content : Value (VV)
```

- And four messages to notify the following events « detection », « VVa^VVar », « Regulation_Decision » and « Engine_Control »:

```
Message_3 :
Source : ECU1
Destination : ECU1
Content : Value (Detection)
Message_4 :
Source : ECU1
Destination : ECU1
Content : Value (VVa ^ VVar)
Message_5 :
Source : ECU2
Destination : ECU1
Content : Value (Regulation_Decision)
Message_6 :
Source : ECU2
Destination : ECU1
Content : Value(Engine_Control)
```

A. *Property verification results for SDK*

By parsing the AADL architecture description, we verified that the description of the architecture meets the properties 1, 2, 3 and 4 of functional-architectural diagnosability. In the following, we check properties 5 and 6, they require a deeper analysis.

Property 5 verification:

This property is «TRUE" when the available time is sufficient for a diagnoser (or sub-function of diagnoser, in the case of decentralized diagnosis).

The proposed diagnoser for SDK function is indeed a general diagnoser for the entire vehicle. The objective is to ensure that SDK function will always be diagnosable even when it is implemented on the same computing unit that the diagnoser.

After a co-simulation of one minute of system operation, COSITA tool displays an interface that allows opening co-simulation trace file(s) in order to verify temporal availability for a diagnoser on "ECU1" (SDK_computing_unit). Given that:

- The desired frequency of diagnosis is once every 4200ns = 4200/25= 168 clock cycles,
- The duration of the diagnosis process is estimated at 160 cycles = 160*25ns= 4000ns
- And the access time to the computing unit for read or write is estimated at 10 ns.

COSITA opens trace files, allow entering information about the diagnoser process and the implemented function(s) onto the component that we want to analyze. Then, COSITA conducts an analysis of the temporal availability of the selected component regarding the desired and entered parameters, based on co-simulation trace files analysis.

With the parameters we have, the module OBSAN concluded that the analyzed component (ECU1) is not available to execute a diagnosis process with the desired criteria. The result is negative for the duration of the execution, because the function analyzed is periodic and the intended diagnosis process is periodic also. Hence, if the result is negative for a period it is also for all others throughout the simulation (Figure 7).

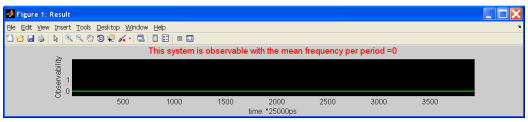

Figure 7. Temporal availability analysis result

Thus, with the proposed features for architecture, property 5 is not satisfied.

Property 6 verification:

To satisfy this property, we have to:

- Check that there is a temporal availability at the system level to implement a diagnoser,
- check the observer accessibility to variables carrying information from the sensors, depending on the functional diagnosability model,
- and check the Reachability of computing units participating in the concerned event.

The last two points of the property 6 are satisfied, and the temporal availability is not satisfied. Thus, property 6 cannot be satisfied.

Since both properties 5 and 6 are not satisfied, the functional-architectural diagnosability is not satisfied. We note that although the SDK function is functionally diagnosable, his proposed software-hardware architecture prevents it from being diagnosed. In other words, if we neglect the architecture constraints at the diagnosability analysis time, the implementation of a diagnoser in the next step may have problems that contradict the positive result of diagnosability.

Our approach proposes in this case to make changes to the architecture description and the model of hardware-software architecture in order to make it diagnosable and not prevent it from being diagnosed. This amounts to the fact that the diagnosability analysis is done at the system design step and the necessary changes to improve the system must be done on models.

B. *Return on design*

After property verification, we should find a solution at the design level of hardware-software architecture in order to satisfy "Temporal availability". We propose in this case, two different solutions:

- The first one is to use a computing unit more powerful instead of ECU1 (SDK_computing_unit) provided for the implementation of SDK and the diagnoser.
- The second solution is to add a computing unit having the same frequency of ECU1 (SDK_computing_unit) and ECU2 (Engine_computing_unit) in order to implement onto it the diagnoser.

Below, we test the two proposed solutions by estimating the cost of each case.

1) *First proposal for modifying the architecture*

Taking a more powerful computing unit of 80MHz for example (taken off the shelf), the duration of the clock cycle is $\frac{1}{80*10^6}$s = $12.5.10^{-9}$ s = 12.5 ns.

As the “SDK ()” method is executable in 16 clock cycles, it is then executable on this computing unit in 16*12.5 = 200ns. Similarly, the diagnoser is executable on 160 clock cycles, it is then executable on this new computing unit on 160*12.5 = 2000ns.

We presume that the desired frequency of diagnosis is the same as in the previous configuration of architecture, once every 4200ns.

Hence, for the first solution to test the description and the SystemC model of the architecture remains the same except the clock of ECU1 which must pass to 80MHz:

```
sc_clock clock("clock",12.5,SC_NS);
```

At checking properties of the architecture, the results remain the same for the property 1.1, property 3 and property 4; because the description of the architecture has changed at the level of attribute "microprocessor" of SDK_computing_unit:

```
system implementation SDK_computing_unit.imp
  subcomponents
    ……;
    Proc : processor MPC563.Motorola {Clock_Period
=> 12.5 ns ;};
    …..;
end SDK_computing_unit.imp;
```

Thus, property 2 was re-checked by parsing AADL architecture description.

Property 2 verification result:

∀d ∈ D, ∀f ∈ F_Diag,

Executability (f,d)=1

At checking functions-architecture interaction, we co-simulate this configuration of hardware-software architecture with the same SDK Simulink model and the same inputs values as the previous configuration, which reproduces the same result in the execution of the SDK function.

Property 5 verification result:

The access time to this computing unit for read/write is estimated at 10 ns. The desired frequency of diagnosis is the same as in the previous configuration of architecture, once every 4200 ns, equivalent to $\frac{4200}{12.5}$ = 336 clock cycles. The duration of the diagnoser process is the same 160 cycles equal to 12.5*160 = 2000ns.

With the new entered parameters, Temporal availability analysis of the 80MHz computing unit becomes true.

Property 6 verification result:

The property "Temporal availability is satisfied, the property" Observability "is then also satisfied, because his other two sub-points "Reachability" and "Accessibility" are already satisfied.

With the proposed changes, the architecture becomes diagnosable from a functional-architectural point of view.

2) Second proposal for modifying the architecture

The second proposal solution is to a computing unit to the architecture, having the same power of « SDK_computing_unit» and « Engine_computing_unit » (40MHz) in order to implement onto it only the diagnose (Figure 8).

The 40MHz computing unit has a clock cycle of $\frac{1}{40*10^6}$ s = 25.10^{-9} s = 25 ns.

Figure 8. Configuration of the hardware-software architecture with an additional computing unit

Messages destinations change following the new description of the architecture :

- The two messages containing state variables values,

```
Message_1 :
Source : ECU1
Destination : ECU3
Content : Value(RV)
Message_2 :
Source : ECU1
Destination : ECU3
Content : Value (VV)
```

- And four messages to notify the following events « detection », « VVa^VVar », « Regulation_Decision » and « Engine_Control »:

```
Message_3 :
Source : ECU1
Destination : ECU3
Content : Value (Detection)
Message_4 :
Source : ECU1
Destination : ECU3
Content : Value (VVa ^ VVar)
Message_5 :
Source : ECU2
Destination : ECU3
Content : Value (Regulation_Decision)
Message_6 :
Source : ECU2
Destination : ECU3
Content : Value(Engine_Control)
```

At checking properties of the architecture -of functional-architectural diagnosability- we verify that the system description meets the required properties of the architecture.

Variables values are:

A= {ECU1, ECU2}

B = {ECU1}

C= {ECU1, ECU2}

D_principal= ECU3

D : All computing units (or components) diagnosers

E = {Detection, (Vva^Vvar), Regulation_Decision, Engine_Control}

F : All functions

F_Diag= {ECU3}

M= {Message_1, Message_2, Message_3, Message_4, Message_5, Message_6}
O : The computing unit (or component) containing the observer process
V= {Rv, Vv}

Rule 1 verification **:**
Since the diagnosis framework used is centralized, we have to check properties 1.1, 2, 3, 4, 5, 6 and 7. After parsing AADL description, properties 1.1, 2, 3and 4 are satisfied. Then, to verify properties 5 and 6, we have first integrated in the SystemC model of the architecture the model of the new computing unit (and its link through the CAN bus to ECU1 ECU2).
Since our objective is not interested in achieving a diagnoser, we have incorporated a simplified diagnostic process in the model, which reads all values sent by SDK_computing_unit and Engine_computing_unit. We co-simulate this configuration of hardware-software architecture with the same Simulink model of SDK function and the same input values as the previous configuration. We get the same result in the execution of the SDK function on ECU1 and ECU2.
Property 5 verification:
The access time to this computing for read / write is estimated to be 10ns. The desired frequency of diagnosis is the same as in the previous configuration of architecture, one time per 4200ns which is equivalent to $\frac{4200}{25}$ =168 clock cycles. The duration of the diagnoser process is the same 160 cycles equal to 25*160= 4000 ns. With the new parameters entered, the analysis of the temporal availability of the 40MHz additional computing unit becomes favorable.
Property 6 verification:
The property "Temporal availability" is satisfied, the property "Observability" is then also satisfied, because his other two points needed "Reachability" and "Accessibility" are already satisfied. With the proposed changes, the architecture becomes diagnosticable in the functional-architectural point of view.

C. Comparison of two solutions

The two solutions we have proposed lead to two different configurations of diagnosable architecture. The major difference between the two solutions is the cost; a difference estimated to be approximately 1800 Euros (there are hidden costs of engineering and technical validation) offers greater speed of execution (Table 1).

Table 1. Summary of advantages and disadvantages of both solutions

Solution	ECU	Characteristics	Diagnosability	Cost
1	ECU1 : 80MHz	SDK run: 200ns Diagnosis run: 2000ns	Yes	500 Euros extra compared to a 40MHz computing unit
	ECU2 : 40MHz			
2	ECU1 : 40MHz	SDK run: 400ns	Yes	2000 euros (price of a computing unit) + 300 euros (price of connecting cables)
	ECU2 : 40MHz			
	ECU3 : 40MHz	Diagnosis run : 4000ns		

In the case of the system we have analyzed, we did not need a faster execution of the SDK function or of the diagnosis function; our attention is focused on diagnosable hardware-software architecture. Thus, the first solution is more reasonable.

VII. CONCLUSIONS AND PERSPECTIVES

According to the literature, taking into account the characteristics of the hardware architecture -often considered as "constraints" of implementation- for the analysis of diagnosability is very original. The method we developed to compare the model of architecture (expressed in AADL) with the classical functional model of diagnosability is unusual.

Properties of the architecture and functions-architecture interaction have been defined to analyze the functional-architectural diagnosability. During construction of our approach, these properties were considered as indicators for the verification of the diagnosability of an embedded system. However, these properties have gradually become requirements for iterative design of hardware-software architecture, including registering and verification of diagnosability as the major step.

As future works, we aim at automating the AADL code parsing following functional-architectural properties. We aim also at resuming the development of our prototype COSITA to refine and finalize the automatic generation of scenarios. We must also build more bridges to existing tools, both those used for capturing requirements and for the verification of temporal constraints.

REFERENCES

[1] J.C. Laprie, *Dependability: Basic Concepts and Terminology* Springer-Verlag, 1992. ISBN 0387822968
[2] W. Hamscher, et al, Readings in model-based diagnosis. Morgan Kaufmann Publishers Inc., ISBN: 1-55860-249-6, 1992.
[3] X. Pucel, A unified point of view on diagnosability, 2008
[4] M. Sampath, et al, Diagnosability of discrete-event systems. *IEEE transactions On Automatic Control 9(40)*, p-p: 1555-1575, 1995.
[5] M. Khlif, M. Shawky, Observability Checking to Enhance Diagnosis of Real Time Electronic Systems. *DS-RT 2008. The 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications*. Vancouver, British Columbia, Canada, 2008.
[6] http://www.aadl.info/aadl/currentsite/
[7] J. Rouillard, Lire & Comprendre VHDL & AMS. *Lulu éditeur*, ISBN 978-1-4092-2787-8, 2008.
[8] M. Khlif, O. Tahan, M. Shawky, CO-SImulation Trace Analysis (COSITA) Tool for Vehicle Electronic Architecture Diagnosability Analysis. University of California, San Diego, CA, USA, 2010.
[9] R. Wilhelm. The Worst-Case Execution Time Problem— Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems (TECS)*, Vol 7, Issue 3, 2008.
[10] Boland, J.F., Thiebeault, C. And Zilic, Z.; *Using MATLAB and Simulink in a SystemC verification environment*; 2nd North American SystemC User's Group. Santa Clara, CA, USA. 2004.
[11] Varona-Gomez, R. And Villar, E.; *AADL Simulation and Performance Analysis in SystemC*; Proceedings of the 14th IEEE Conference on Engineering of Complex Computer System, 2009
[12] X. Claeys, and al, "Chauffeur Assistant Functions," *Report restricted to RENAULT TRUCKS*, European Contract number IST-1999-10048, Lyon, FRANCE, 2003
[13] Anon. *Software Considerations in Airborne Systems and Equipment Certification*, DO-178B RTCA, Washington D.C. 1992
[14] Chilensky, J.J. and Miller, S.P. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 1994.