Heterogeneous Redundancy Analysis based on Component Dynamic Fault Trees

Jose Ignacio Aizpurua[a*], Eñaut Muxika[a], Ferdinando Chiacchio[b], and Gabriele Manno[c]
[a] University of Mondragon, Mondragon, Spain
[b] University of Catania, Catania, Italy
[c] Strategic Research & Innovation DNV GL, Høvik, Norway

Abstract: The aggregation of hardware components to add recovery capabilities to a system function may result in high costs. Instead of adding redundancies with homogeneous nature aimed at providing recovery capabilities to a predefined system function, there is room in some scenarios to take advantage of over-dimensioning design decisions and overlapping structural functions using heterogeneous redundancies: components that, besides performing their primary intended design function, restore compatible functions of other components.

In this work, we present a methodology to evaluate systematically the effect of failures of alternative redundancy and reconfiguration strategies, fault detection, and communication implementations on system dependability. To this end, a modeling approach called Generic Dependability Evaluation Model and its probabilistic analysis paradigm using Component Dynamic Fault Trees are presented. Application to a railway example is presented showing tradeoffs between dependability and cost when deciding to implement possible redundancy and reconfiguration strategies. Finally, details of the experiment prototype implemented using real railway communication elements are described so as to validate the design concepts treated throughout the paper.

Keywords: Heterogeneous redundancies, Dependability, Design methodology, Monte Carlo simulations, Cost reduction.

1. INTRODUCTION

Traditional design strategies to improve fault tolerance of a system are based on the replication of hardware components in redundancy configurations. Generally the nature of the backup components is homogeneous, i.e., they provide recovery capabilities to a predefined system function; accordingly they are known as homogeneous redundancies.

However, in some scenarios, it is possible to make use of heterogeneous redundancies, consisting of components that, besides performing their primary intended design function, are able to restore compatible functionality of other components. This is the case of Massively Networked Scenarios (MNS), systems characterized by several replicas of system functions throughout the physical structure, e.g., a train has replicated functions throughout its cars; a building has replicated functions throughout its floors and rooms. Nowadays, in such architectures, the comparison between homogeneous and heterogeneous redundancies is gaining the interest of researchers and industrial stakeholders. In fact, design strategies based on heterogeneous redundancies have shown potential to improve system dependability cost-effectively [1, 2, 3].

In previous work [4], an adaptive dependable design methodology, the D3H2 (aDaptive Dependable Design for systems with Homogeneous and Heterogeneous redundancies), for the dependability

Acronyms & Abbreviations: (C)DFT: (Component) Dynamic Fault Tree; FD_R_SF: Fault detection of the R_SF; FD_SF: Fault Detection of the SF; GDEM: Generic Dependability Evaluation Model; (E)GFM: (Extended) Generic Functional Model; MF: Main Function; MVB: Multi-function Vehicle Bus; PL: Physical Location; PU: Processing Unit; R_SF: Reconfiguration of the SF; SF: Subfunction.

* Contact E-mail address: jiaizpurua@mondragon.edu

assessment of MNS was presented. Applying modeling and analysis techniques, the methodology identifies heterogeneous redundancies systematically; integrates redundancies in the *extended HW/SW architecture* supporting necessary functions and implementations including reconfiguration, fault detection, and communication; and performs dependability and cost analyses of the *extended HW/SW architecture*. In this paper, we present an extension of the methodology overcoming previous static logic limitations by means of Generic Dependability Evaluation Model (GDEM) and Component Dynamic Fault Trees (CDFTs), and quantify the effect of the failure of redundancy and reconfiguration strategies on the *extended HW/SW architecture*. The GDEM defines failure relationships between system functions and implementations and allows identifying systematically the combinations of faults that lead the *extended HW/SW architecture* to fail; while the CDFT enables the probabilistic assessment of the GDEM. Besides, importance measurements are performed to obtain robustness indicators of design strategies. To test the feasibility of the approach, its key design concepts are implemented in a practical case of the railway industry: reconfiguration capabilities have been added to hardware train network components to reuse already existing elements.

Related Work: The evaluation of the influence of design decisions on dependability and cost is an ongoing research challenge. While many works have concentrated on addressing the influence of homogeneous redundancies on system dependability and cost, approaches focusing on the evaluation of the influence of heterogeneous redundancies on system dependability are scarce.

The concept of heterogeneous redundancies have been addressed in the literature with different names but with the same underlying design goal: reuse of system components to provide a compatible functionality. Shelton and Koopman worked on the concept of functional alternatives [1]; Wysocki and Debouk presented a methodology for assessing architectures using shared redundancies [2]; and Adler et al. presented a methodological support for characterizing an adaptation model while meeting availability-cost requirements [3]. However, to fully exploit the potential of heterogeneous redundancies without incurring in additional penalties on the system architecture, there exist assumptions and activities that should be addressed: (1) identification of heterogeneous redundancies should be done systematically rather than relying only on the ability of the designer; (2) the use of homogeneous/heterogeneous redundancies in MNS requires fitting the system with health management implementations, i.e., fault detection (FD) and reconfiguration (R). Consequently, alternative health management strategies and their influence on dependability should be addressed so as to avoid unexpected consequences. For further details and discussion in this area see [5].

Contribution and overview of the paper: as a result, with respect to relevant approaches, we propose a methodology that: (1) identifies systematically heterogeneous redundancies and integrates them in the system architecture; (2) constructs alternative system architectures comprehending different redundancy and reconfiguration strategies; and (3) performs the dependability assessment of system architectures systematically and exhaustively with the goal of extracting design indicators and identifying weaknesses and strengths of the analyzed system architectures. The remainder of this paper is organized as follows: Section 2 overviews the D3H2 methodology and introduces the application example; Section 3 describes in detail the dependability analysis within the D3H2 methodology, and its application to a railway example; Section 4 validates D3H2's design concepts in a experiment prototype; and finally, Section 5 presents conclusions and our future research goals.

2. D3H2 METHODOLOGY: OVERVIEW & APPLICATION EXAMPLE

In this section we present an overview of the main activities of the D3H2 methodology [4] and apply them to an application example.

2.1. Overview

The D3H2 methodology characterizes a system of interest as a set of HW, SW, and communication components taking into account their interfaces and provided functionality. The main goal of D3H2 methodology is to create a system architecture that meets dependability and cost requirements, and

evaluate the influence of redundancy (homogeneous, heterogeneous) and reconfiguration (centralized, distributed) strategies on system dependability and cost systematically (see Figure 1). The methodology integrates the following main modeling and analysis activities:

(1) Construction of the *functional model* based on the next *tokens:* (1) *Main Function* (MF) (e.g., Temperature Control); (2) the *Physical Location* (PL) in which the MF is performed; (3) necessary set of input (I), control (C) and output (O) S*ubfunctions* (SFs) to perform the MF (e.g., Temperature Measurement); and (4) *implementations* of the SF characterized by the HW, SW, and communication resources they use and ordered with respect to their implementation priority.

(2) *Compatibility analysis*: systematic identification of heterogeneous redundancies based on the physical location of the SF and the compatibility of the SF they perform; and extraction of *reconfiguration strategies* and priorities.

(3) Construction of the *extended HW/SW architecture* by adding health management implementations to the *preliminary HW/SW architecture*, i.e., fault detection (FD_SF), reconfiguration (R_SF), and reconfiguration's fault detection (FD_R_SF) implementations.

(4) And finally, *dependability and cost analysis* of the *extended HW/SW architecture* to validate dependability and cost requirements. The ins and outs of these activities are described in [4].

Figure 1: D3H2 Methodology [4]

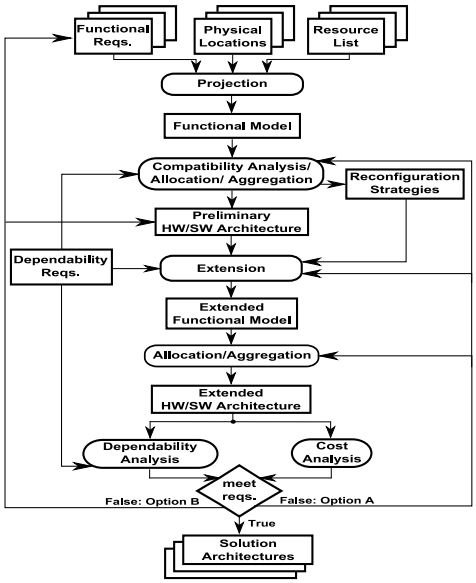
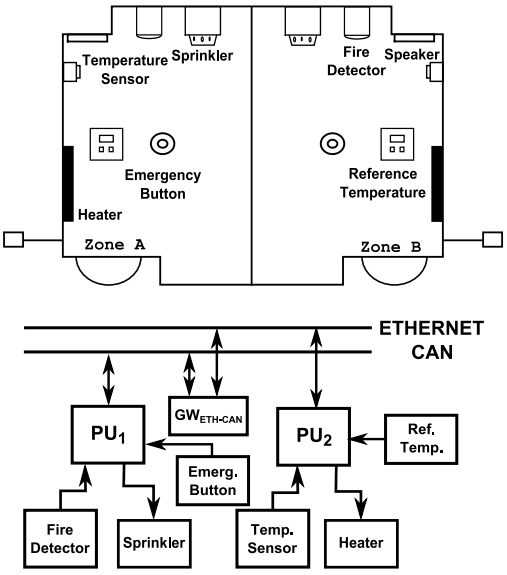


Figure 2: Train Car and Network Structure



2.2. Application Example

In this subsection, we apply and describe the D3H2's main activities in an application example of the railway industry. Namely, we concentrate on analyzing Fire Protection Main Function and possible heterogeneous redundancies arising from the analysis of a train car.

System: in the studied train car fire detectors are distributed strategically to detect a fire. Usually, there is a fire detector (smoke sensor) for each train car's compartment (see $Zone_A$ - $Zone_B$ in Figure 2) and an emergency button for passengers. Potential fire conditions are validated by a fire control algorithm and if confirmed, sprinklers neutralize the hazardous situation.

Functional Model: among different Main Functions performed in a train car (cf. Table 1), the functional model of the Fire Protection and part of the Main Functions located at compatible physical location are described (Fire Protection and Temperature Control). There exists other Main Functions located at the same physical location, but for the sake of clarity, we have not taken them into account.

Compatibility Analysis & Reconfiguration Strategies: the compatibility analysis is based on 2 compatibility cases: (1) implementation of the same SF in compatible PLs; and (2) different SF's I/O implementations located at compatible PL. The first case is performed automatically comparing SFs

and PL, while the second case requires evaluating if they may fulfill additional compatible SFs. From the second case, the compatibility analysis automatically suggests a list of implementations located at a Fire Protection's compatible PL ($Train.Car_1.Zone_A$) and the designer intuitively finds a temperature sensor able to indicate the presence of fire using temperature value thresholds (see grey cells at the functional model in Table 1) (see [4] for more information of the compatibility analysis).

Extended Functional Model (EFM): based on design decisions, the EFM is constructed by adding necessary fault detection and reconfiguration mechanisms to the SFs with redundant implementations to evaluate their influence on dependability and cost. For instance, the *extended HW/SW architecture* shown in Table 2 is based on a model with single FD_SF, duplicated R_SF implementations, and each R_SF implementation with its FD_R_SF implementation.

Table 1: Functional Model

MF	PL	SF	Resources
Fire Protection	Train. Car1. $Zone_A$	User Emerg. Signal (UES)	Emergency Button, SW_{UES}, PU1
		Fire Detection	Fire Detector, $SW_{FireDet}$, PU1
		Fire Control	UES, Fire Detection, PU1, $SW_{FireControl}$
		Fire Extinction	Fire Control, PU1, $SW_{FireExtinction}$, Sprinkler
Temp. Control	Train. Car1. $Zone_A$	Temperature Measurement	Temperature Sensor, SW_{Temp}, PU2
		…	…

- **Heterogeneous Redundancies (Semi-Automatic)**
- **Health Monitoring Implementations (Systematic)**
  - **FD_SF, R_SF and FD_R_SF**

Table 2: Extended HW/SW Architecture

MF	PL	SF	Resources
Fire Prot.	Train. Car1. $Zone_A$	UES	Emergency Button, SW_{UES}, PU1
		Fire Detection	Fire Detector, $SW_{FireDet}$, PU1
			Temperature Sensor, $SW_{FireDet}$, PU2, Comm. CAN, Comm. ETH, $GW_{ETH-CAN}$
		FD_FireDet	PU1, $SW_{FD\_FireDet}$, Comm. CAN, Comm. ETH, $GW_{ETH-CAN}$
		R_FireDet	PU1, $SW_{R\_FireDet}$, Comm. CAN
		R_FireDet	PU2, $SW_{R\_FireDet}$, Comm. CAN, Comm. ETH., $GW_{ETH-CAN}$
		FD_R_FireDet	PU2, $SW_{FD\_R\_FireDet}$, Comm. CAN, Comm. ETH, $GW_{ETH-CAN}$
		FD_R_FireDet	PU1, $SW_{FD\_R\_FireDet}$, Comm. CAN, Comm. ETH, $GW_{ETH-CAN}$
		Fire Control	UES, Fire Detection, PU1, $SW_{FireControl}$, Comm. CAN, Comm. ETH, $GW_{ETH-CAN}$
		Fire Extinction	Fire Control, $SW_{FireExtinction}$, PU1, Sprinkler

As for the dependability analysis, in [4] we focused on automating all the D3H2 activities using the Component Fault Tree paradigm [6], keeping the traceability between the *dependability model* and *extended HW/SW architecture* as manageable as possible. However, so as to adhere to the *extended HW/SW architecture*'s dynamic failure logic, it is necessary an approach which captures systems dynamic failure logic and does not hamper the readability of the *dependability model*. Hence, to address these goals, Section 3 defines compositional Generic Dependability Evaluation Model (GDEM) and Component Dynamic Fault Tree (CDFT) paradigms.

3. D3H2 METHODOLOGY: DEPENDABILITY ANALYSIS

The key introductory concepts for dependability analysis are defined in § 3.1. Then, the GDEM (cf. § 3.2) and its analysis approach based on CDFTs (cf. § 3.3) are defined.

3.1. Concepts and Notation

The objective of the GDEM is the generic, systematic and complete failure modeling of *extended HW/SW architectures.* The failure model of the *extended HW/SW architectures* comprehends the possible failure modes of its implementations: FD implementations (FD_SF, FD_R_SF) fail in omission (O) when it does not detect a failure when it occurs and false positive (FP) when it detects a failure when it does not exist; the reconfiguration (R) implementation fails in omission when it fails to reconfigure an implementation when it is required; and failure of SF implementations cover omission and wrong value failure modes.

The failures of all system subfunction implementations (SF, FD_SF, R_SF, FD_R_SF) are defined at the implementation level (i.e., [MF].[PL].[SF].[Impl] Failure) according to the failures of the implementation's resources. Combining implementation level failures, SF level failures are defined

systematically ([MF].[PL].[SF] Failure). For the sake of clarity, in subsequent characterizations we omit the generic common part ([MF].[PL]). Table 3 defines the notations of the failure events and working events according to their SF and failure modes (omission and false positive).

Table 3: Notation of Failure and Working Events

Notation	Failure Logic
F_X	X failure
F_{SF}	[SF] failure
F_{SF_i}	[SF].[Impl$_i$] failure
F_{FD_i}	[FD_SF].[Impl$_i$] failure
F_{FD}	[FD_SF] failure
$F_{FD\ FP}$	[FD_SF] false positive
$F_{SF_i\ FP}$	[SF].[Impl$_i$] failure or FP = **OR**(F_{SF_i}, $F_{FD\ FP}$)
$F_{FD_i\ O}$	[FD_SF].[Impl$_i$] omission

Notation	Failure/Working Logic
F_R	[R_SF] failure
$F_{R_i\ O}$	[R_SF].[Impl$_i$] omission
$F_{FD\_R_i\ FP}$	[FD_{[R_SF].[Impl$_i$]}] false positive
$F_{R_i\ O/FP}$	[R_SF].[Impl$_i$] omission or FP = **OR**($F_{R_i\ O}$, $F_{FD\_R_i\ FP}$)
$F_{FD\_R_i}$	[FD_{[R_SF].[Impl$_i$]}] failure
W_X	X working
W_{SF_i}	[SF].[Impl$_i$] working = **NOT**(F_{SF_i})

The stochastic failure characterization of each resource is characterized randomly sampling the failure times according to their cumulative probability distribution functions (CDFs) along the system lifetime. The methodology supports any CDFs, but for the sake of simplicity, without losing the generality of the approach, in subsequent probabilistic characterizations exponential failure distributions are assumed. Hence, the failure characterization of system resources is defined according to their failure rates (λ_{Res}). The failure characterization of a SF's *i*-th implementation ([SF].[Impl$_i$]) with *N* resources is specified as follows:

$$F_{SF_i} = \mathbf{OR}(F_{Res_1}, F_{Res_2}, \ldots, F_{Res_N}) \quad (1)$$

The same characterization holds for FD_SF, R_SF, and FD_R_SF implementations.

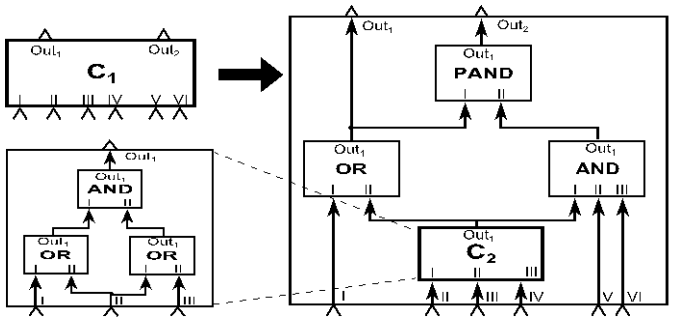
3.2. Component Dynamic Fault Trees (CDFT)

To quantify the failure probability of the GDEM, we have analyzed existing dynamic and compositional fault tree paradigms looking for the following characteristics: (1) component-based failure characterization: embed the failure logic of a set of related events/components and (re)use it where needed instead of characterizing the system failure behavior in a single flat model; (2) dynamic gates: capture the system failure logic accounting for the time-ordered events; (3) NOT gates: address the influence of functional (NOT failed) events; (4) support for any probability density function; (5) possibility of modeling repeated Basic Events (BEs); and (6) repeated components.

The integration of static fault trees and compositional characterization is not new: Component Fault Trees [6] addressed this concept prominently. However, to the best of our knowledge, there is no approach which addresses explicitly the integration of DFTs and component oriented characterization. There exists combinations of combinatorial and state-based approaches which do address the compositional characterization (e.g., [7]), but state-based approaches are not considered in this work in order to avoid possible state explosion and manageability issues.

Addressing all the aforementioned characteristics, we define the *concept* of Component Dynamic Fault Trees (CDFTs): while a BE characterizes self-contained simple failure logic, a component encloses any-complexity failure logic (with possibly multiple I/O dependencies) specified using BEs, gates, and even further components. The CDFT paradigm makes it possible to embed in a component the dynamic failure logic of a (sub)system and (re)use it where needed addressing repeated components and repeated BEs. Figure 3 characterizes a hypothetical CDFT model with repeated components (C_2) and CDFT gates. Each component (C_1, C_2) may have as inputs further gates, or (repeated) BEs and/or other components. The behavior of the CDFT gates are characterized according to its inputs events (A, B), which are extendible to an arbitrary number of input events.

Figure 3: Component Dynamic Fault Tree Overview



Gate Notation	Component Behaviour
Y=**AND**(A, B)	If A fails and B fails, then Y fails
Y=**OR**(A, B)	If A fails or B fails, then Y fails
Y=**PAND**(A, B)	If A fails before the failure of B or at the same time, then Y fails
Y=**NOT**(A)	If A doesn't fail, then Y fails

To *implement* the CDFT paradigm and analyze the (un)reliability of a system, Monte Carlo simulations are performed on the system's CDFT structure calculating the time to failure of BEs according to their cumulative probability distribution function. Connected gates/components use this information to determine their outcome (functional or failed state). When a failure at the output of a gate/component occurs, the failure time information is passed to the next gate/component so that the system's dynamic failure logic is tracked from BEs to system-level Top Event (TE). The logic of CDFT gates comprehend combinatorial (AND, OR, NOT) and time-ordered (PAND) failure logic. To analyze CDFTs, the MatCarloRe tool [8] has been extended with NOT gates.

3.3. Generic Dependability Evaluation Model (GDEM)

The GDEM defines the generic but implementable dependability evaluation algorithm, defining the dynamic failure behavior of systems which use fault detection and reconfiguration implementations covering all possible failure situations for the specified HW/SW architectures. It allows evaluating design decisions consequence on system dependability. Resulting equations characterize the failure of such systems compositionally so that the failure logic is kept clear for complex systems.

To this end, the GDEM characterizes combinations of SF's implementation failures that prevent the *extended HW/SW architecture* from performing its intended MF. The failure of any SF necessary for a MF provokes its immediate failure. Hence, from this point onwards we will analyze the failure of a SF. The subfunction will fail when all its implementations have failed ($F_{\text{All Impl.}}$), an implementation fails and reconfiguration does not happen (failure unresolved, $F_{\text{Unresolved}}$), or its input dependencies ($F_{\text{Dependencies}}$) have failed (cf. Equation 2):

$$F_{SF} = \mathbf{OR}(F_{\text{All Impl.}}, F_{\text{Unresolved}}, F_{\text{Dependencies}}) \quad (2)$$

Assuming that there exist M implementations of the subfunction, the $F_{\text{All Impl.}}$ event happens when each implementation fails or is detected as failed (false positive):

$$F_{\text{All Impl.}} = \mathbf{AND}(F_{SF_1\,FP}, ..., F_{SF_M\,FP}) \quad (3)$$

The failure unresolved ($F_{\text{Unresolved}}$) occurs when the working implementation fails and either the fault is not detected or the reconfiguration itself fails. For each implementation there are different failure unresolved events ($F_{\text{Unr. Impl}_i}$) because each implementation may have different failure probabilities, however, note that the last implementation's failure probability can not be solved:

$$F_{\text{Unresolved}} = \mathbf{OR}(F_{\text{Unr. Impl}_1}, ..., F_{\text{Unr. Impl}_{M-1}}) \quad (4)$$

To define $F_{\text{Unr. Impl}_i}$, let us introduce two new events. The first event occurs when the i-th implementation of the subfunction fails and the reconfiguration has failed but after successfully reconfiguring previous i-1 implementations (reconfiguration sequence failure, $F_{\text{R Seq.}_i}$). Assuming $F_{SF_{1..i-1}\,FP} = \mathbf{AND}(F_{SF_1\,FP}, ..., F_{SF_{i-1}\,FP})$ indicates the failure or false positive from 1 to i-1 implementations:

$$F_{\text{R Seq.}_i} = \mathbf{PAND}(F_{SF_{1..i-1}\,FP}, F_R, F_{SF_i\,FP}) \quad (5)$$

The second event occurs when the *i*-th implementation of the SF fails and the fault detection of the SF has failed but after detecting correctly previous *i*-1 implementations failures (fault detection sequence failure, $F_{FD\ Seq._i}$). Note that fault detection's false positive and omission failures are mutually exclusive:

$$F_{FD\ Seq._i} = \mathbf{PAND}(F_{SF_{1..i-1}}, F_{FD,}\ F_{SF_i}) \quad (6)$$

Due to the characterization of time-ordered failures, Equations 5 and 6 can not be further simplified. Accordingly, *i*-th implementation's failure unresolved event ($F_{Unr.\ Impl_i}$) occurs when either the fault detection sequence ($F_{FD\ Seq._i}$) or the reconfiguration sequence ($F_{R\ Seq._i}$) fails:

$$F_{Unr.\ Impl_i} = \mathbf{OR}(F_{R\ Seq._i}, F_{FD\ Seq._i}) \quad (7)$$

Dependencies address Input (I) and Control (C) subfunctions influence on control and Output (O) SFs respectively. Control SF failure impacts the output SF failure directly (C→O); and the influence of input SF on control SF depends if the system's control configuration is operating in Closed Loop (C_CL) or Open Loop (C_OL):

$$F_{Dependencies} = \mathbf{OR}(F_{Dep.\ C\_CL}, F_{Dep.\ C\_OL}) \quad (8)$$

Assuming that $W_{C\_X} = \mathbf{OR}(W_{C\_X_1}, \ldots, W_{C\_X_Q})$ means that all *Q* implementations of C_X subfunction are working, equations in 9 describe the different input subfunctions that affect each control configuration (I_CL→C_CL, I_OL→C_OL). Usually, the $F_{Dep.\ C\_OL}$ event will not happen because the open loop control generally does not have input dependencies:

$$F_{Dep.\ C\_CL} = \mathbf{AND}(W_{C\_CL}, F_{I\_CL}) \qquad F_{Dep.\ C\_OL} = \mathbf{AND}(W_{C\_OL}, F_{I\_OL}) \quad (9)$$

The reconfiguration failure is a special subfunction and therefore F_R is developed like Equation 2, except that there are no additional dependencies:

$$F_R = \mathbf{OR}(F_{All\ R\ Impl.}, F_{R\ Unresolved}) \quad (10)$$

$F_{All\ R\ Impl.}$ indicates the failure of all reconfiguration implementations, and $F_{R\ Unresolved}$ designates the reconfiguration's failure unresolved condition. Assuming *P* reconfiguration implementations:

$$F_{All\ R\ Impl.} = \mathbf{AND}(F_{R_1\ O/FP}, \ldots, F_{R_P\ O/FP}) \quad (11)$$

$F_{R\ Unresolved}$ event happens when *P*-1 implementations of the FD_R subfunctions fail:

$$F_{R\ Unresolved} = \mathbf{AND}(F_{FD\_R_1}, \ldots, F_{FD\_R_{P-1}}) \quad (12)$$

Equation 12 boils down to our design choice: all reconfiguration's fault detection implementations (FD_R_SF) are active and homogeneous redundancies (heartbeat implementation). Accordingly, the false positive occurs when all FD_R_SF implementations raise the false positive condition simultaneously. Although the system may operate correctly when a false positive occurs, it has to assume that the information provided by the fault detection implementation is correct, since there is no mechanism to detect the incorrect operation of fault detection.

The fault detection failure F_{FD} depends on the operation of the destination subfunction (SF_{DEST}), because the FD implementation is located at the same PU. Hence, F_{SF_DEST} influences directly F_{FD}. When the FD implementation fails, the change of SF_{DEST}'s implementation determines its reconfiguration. We assume that the change of destination SF's implementation activates the corresponding FD implementation and the previous one is deactivated. Equation 13 describes the FD_SF failure case when FD_SF has *K* implementations:

$$F_{FD} = \mathbf{OR}(F_{FD\_Dest_1}, \ldots, F_{FD\_Dest_K}) \quad (13)$$

As for the *i*-th fault detection implementation's failure ($F_{FD\_Dest_i}$), it happens when the first 1 to *i*-1 implementations of the destination SF fail and reconfigure correctly ($F_{SF\_DEST_{1..i-1}}$), and then the *i*-th implementation of the fault detection or destination SF fails:

$$F_{FD\_Dest_i} = \mathbf{PAND}(F_{SF\_DEST_{1..i-1}}, \mathbf{OR}(F_{SF\_DEST_i}, F_{FD_i O})) \quad (14)$$

3.4. Experiments

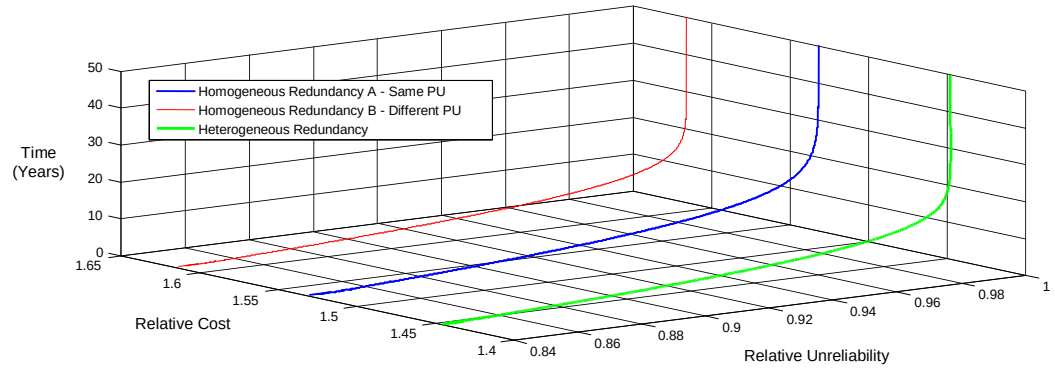
Based on the example described in Section 2, simulations are performed to evaluate the influence of redundancy and reconfiguration strategies on system's (un)reliability and cost. The cost of health monitoring SW components (SW_HM: SW_FD, SW_R, SW_FD_R) is quantified considering their development cost. Regarding their λ values, hypothetical reasonable values are assumed, considering them lower than less critical components' failure rates. The assessment of the reliability and cost of SW components is outside the scope of this work, see [9] for an application on SW reliability methods. Regarding sensor's cost, human cost related with mounting and testing tasks is considered assuming 10 minutes/sensor. PU element characterizes all PU elements, and communication includes CAN and Ethernet communication protocols and their gateway. We are aware that the cost of SW components is greater than adding sensors, but in this example it has been assumed that SW development costs will be paid off in 4 years.

Table 4: Failure Rates & Cost of HW/SW/Communication Components

Component	Fail. Rate (year^{-1})	Cost (€)
SW_HM and SW_FD_FP	1 E-2	80 each
Fire Detector [10] / Temp. Sensor [11]	3.77 E-2 / 1.49 E-2	20 + 60€/hour / -
PU [9]	3.87 E-2	30
Communication and Gateway	5 E-3	200

Figure 4 describes Fire Protection configurations' relative unreliability and cost with respect to the configuration without redundancies described in Table 1. Alternative *extended HW/SW architectures* are analyzed adding a heterogeneous redundancy (*HeR*) (see Table 2) or homogeneous redundancy (*HoR*) to the Fire Detection SF. In case of homogeneous redundancies, the fire detector has been replicated with 2 alternative configurations: connect both fire detectors to PU1 (*HoR A*) or connect each fire detector to a different PU (*HoR B*). As Figure 4 shows, heterogeneous redundancies are more economical than homogeneous redundancies and their unreliability is slightly higher than homogeneous redundancies due to the added mechanisms (SW) to make implementations compatible.

Figure 4: Relative Unreliability and Cost of Fire Protection Configurations (10^6 iterations)



To analyze further the differences between *HoR* and *HeR*, the Failure Criticality Index (FCI) evaluation has been implemented [12] calculating for the component *i*: the ratio between the number system failures caused by component *i* to the total number of system failures. Table 5 shows the impact of the failure of redundancy and reconfiguration (centralized, distributed) implementations on

the Main Function failure. The shown values are the influence on the Fire Protection MF of Fire Detection SF's redundancy ($F_{FireDet_2}$) and its reconfiguration strategies (logic sum of $F_{R\ Seq_1}$ and $F_{R\ Seq_2}$).

Table 5: Failure Criticality Index Values (10^6 iterations)

Causes\Configs.	HoR A – Cen.	HoR B – Cen.	HeR – Cen.	HoR A – Dist.	HoR B – Dist.	HeR – Dist.
Redundancy type	0,339027	0,174606	0,179927	0,276643	0,154960	0,170669
Reconfiguration	0,177554	0,171728	0,174496	0,114232	0,107315	0,106994

FCI values provide indicators about bottleneck influences on system (un)reliability: heterogeneous and homogeneous B redundancies perform better than homogeneous A redundancy configuration due to the bottleneck influence on causing the top event, i.e., PU1 performs as a common cause failure and its failure incurs the simultaneous failure of other SF implementations. The same logic applies to the reconfiguration distribution strategies: distributed reconfiguration implementations perform better than centralized implementations due to the bottleneck influence on system (un)reliability.

4. VALIDATION OF THE D3H2 METHODOLOGY: PROOF OF CONCEPT

To proof the feasibility of the D3H2 methodology in real applications, a key application concept in our methodology has been validated: we have added reconfiguration capabilities to existing hardware train network components to recover the system from failures at runtime using heterogeneous redundancies.

Trains have a standard form of data communication specified in the Train Communication Network (TCN) standard [13]. TCN is a real-time data network comprised of an architecture inter-connecting train vehicles and equipments within a vehicle. The TCN standard specifies Wire Train Bus (WTB) for the inter-connection of vehicles and Multi-function Vehicle Bus (MVB) for intra-vehicle device communication. In this work we focus on the communication within a vehicle using MVB.

MVB operates in master-slave configuration connecting devices in a vehicle. Class 2 or higher devices are considered: intelligent devices participating in the message communication with administration capabilities or connected I/O elements. The master guarantees deterministic medium access managing periodic and sporadic access to the bus. The communication in MVB follows the publisher/subscriber paradigm: a publisher broadcasts variables and this information is distributed to the subscribers. To this end, a traffic store is implemented; each device holds the variables it produces/consumes in a shared memory that is a partial copy of the whole network's distributed database.

As for the implementation of the reconfiguration process, we identify two phases: construction of the *reconfiguration table*, i.e., statically/dynamically determined reconfigurations; and activation of *reconfiguration strategies*, i.e., design-time/runtime reconfigurations. Dynamic resolution of the *reconfiguration table* allows gaining flexibility, but requires exploring the architecture dynamically. For safety and predictability purposes, statically determined reconfiguration design decisions are adopted. Regarding the (de)activation of *reconfiguration strategies*, while design-time reconfigurations reduce design complexity, runtime reconfigurations reduce processing cost and bandwidth overhead activating redundant communication threads exclusively when their need arises.

In a train there are safety-critical functions which must meet hard real-time constraints (e.g., door control) and some of these functions are transmitted through MVB. Besides, other communication protocols coexist in the train; for instance, Ethernet communication protocol transports non-critical infotainment data. Ethernet provides more flexibility to perform architectural modifications at runtime at the expenses of losing predictability with respect to MVB. There exist other communication networks in a train (e.g., CAN), but this proof of concept has been focused on MVB and Ethernet**.**

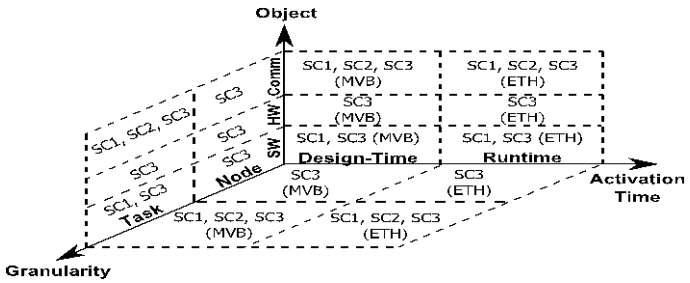
Therefore, the following design decisions have been adopted: MVB has been used for design-time reconfigurations and runtime reconfigurations have been implemented in Ethernet. On one hand, design-time reconfigurations in MVB are performed by assigning reconfiguration routes at design-

time and activating them from the outset. The bandwidth consumption of these redundant communications is constant and their processing is activated solely when their need rise up. On the other hand, (dynamic) runtime modifications in Ethernet are effectuated using UDP communication threads in client-server-like configurations. Communication threads are created and deleted as their need arises, so that the bandwidth and processing costs increase exclusively in case of reconfiguration.

The following reconfiguration scenarios *(SC)* have been tested: (*SC1*) sensor-level heterogeneous redundancy reconfigurations; (*SC2*) communication-level heterogeneous redundancy reconfigurations; and (*SC3*) processing unit-level homogeneous redundancy reconfigurations.

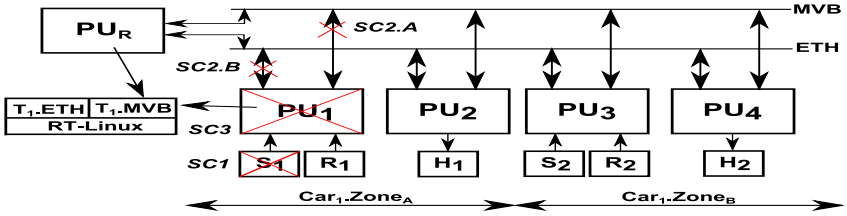
Two additional reconfiguration attributes define the reconfiguration space of these scenarios: reconfiguration *granularity* comprehends node or task level reconfigurations, indicating implementation's reactivation by changing the whole PU and its allocated tasks, or a single task respectively. Reconfiguration *object* addresses SW, HW and communication (Comm.) level reconfigurations: SW reconfigurations modify the SW implementation changing its parameters or structure; HW reconfigurations involve changing the complete HW device; and communication reconfiguration modifies nominal communication routes with alternative ones. Figure 5 describes the reconfiguration space of the tested scenarios: *SC1* deals with sensor failures using communication level reconfigurations, *SC2* switches the communication protocol to handle communication failures, and *SC3* replaces the PU and communication routes to cope with PU failures. For instance, all the scenarios perform task-level and communication-level reconfigurations, but only *SC3* addresses node-level and communication-level reconfigurations.

Figure 5: Reconfiguration Space



SC1: without losing the applicability of the scenario, *SC1* focuses on the example presented in [14]. A train car vehicle may have different compartments (cf. Figure 6 $Zone_A$ - $Zone_B$) and independent Temperature Control Main Function implementations at each compartment. Assume that 2 PUs are connected to perform the Temperature Control in each vehicle's compartment: one PU (PU_1 or PU_3) measures the temperature (*SF_1: TempMeasure*) using a sensor (S_1 or S_2) and gets the reference temperature (*SF_2: RefTemp*) using a reference button (R_1 or R_2), and the other PU (PU_2 or PU_4) acts as a controller (*SF_3: TempControl*) and actuator (*SF_4: Heating*) heating the compartment using heaters (H_1 or H_2). *TempMeasure*'s nominal communication route (*Rt*) in each compartment is as follows: *Rt_1: $S_1 \rightarrow PU_1 \rightarrow ETH \rightarrow PU_2 \rightarrow H_1$; Rt_2: $S_2 \rightarrow PU_3 \rightarrow ETH \rightarrow PU_4 \rightarrow H_2$*. Given that one sensor of any compartment fails, we reuse the already existing one in the same car, but in different compartment.

Figure 6: Reconfiguration Scenarios



Focusing on the reconfiguration of *TempMeasure* at $Car_1.Zone_A$, its value-based fault detection is located in the destination PU_2. When incorrect values are received at PU_2, the reconfiguration implementation (located with the *TempMeasure*'s fault detection), acknowledges to the faulty

component so that it stops sending data. The reconfiguration implementation checks the IP address and the UDP port of the next priority implementation of *TempMeasure* in its *reconfiguration table*, establishing the communication with S_2. This process changes the communication route from Rt_1 to Rt_{12}: $S_2 \rightarrow PU_3 \rightarrow ETH \rightarrow PU_2 \rightarrow H_1$. The design of the devices identified as heterogeneous redundancies enables them to redirect their information to different information sinks dynamically when a reconfiguration signal is received. During the reconfiguration, source and sink PUs synchronize and S_2 continues sending data towards PU_2 until S_1 is restored. Implemented reconfiguration mechanisms are applicable to input SF implementations operating with heterogeneous redundancies (e.g., Fire Protection example). MVB reconfigurations apply the same process, with the difference that Rt_{12} is activated from the outset.

SC2: since a train incorporates different communication protocols, there is room to benefit from heterogeneous redundant communications. Despite bidirectional communications have been implemented between PU_1 and PU_2, for simplicity the following unidirectional routes are considered: Rt_1: $T_1.MVB \rightarrow PU_1 \rightarrow MVB \rightarrow PU_2$; Rt_2: $T_1.ETH \rightarrow PU_1 \rightarrow ETH \rightarrow PU_2$ where *T1.MVB* and *T1.ETH* identify MVB and Ethernet tasks respectively (cf. Figure 6). When a communication link is down, the general communication-level reconfiguration process is as follows: (1) the application located in the sink PU detects the communication failure (time-based fault detection), (2) subsequently, it reconfigures itself creating a server to continue receiving data using the operating communication protocol, and (3) it informs the source PU about the communication failure; (4) finally, the source PU is also reconfigured switching from the faulty to the operative communication. Hence, when MVB is disconnected (*SC2.A*, cf. Figure 6), UDP communication threads are created dynamically to continue sending MVB data via Ethernet changing communication routes from Rt_1 to Rt_{12}, where Rt_{12}: $T_1.MVB \rightarrow PU_1 \rightarrow ETH \rightarrow PU_2$; and vice versa, when Ethernet is disconnected (*SC2.B*, cf. Figure 6) the communication route is changed from Rt_2 to Rt_{22}, where Rt_{22}: $T_1.ETH \rightarrow PU_1 \rightarrow MVB \rightarrow PU_2$.

SC3: point to point unidirectional communication from PU_1 to PU_2 is considered with the next communication routes: Rt_1: $T_1.MVB \rightarrow PU_1 \rightarrow MVB \rightarrow PU_2$; Rt_2: $T_1.ETH \rightarrow PU_1 \rightarrow ETH \rightarrow PU_2$. The tasks that PU_1 is performing are rearranged in another compatible PU to deal with PU_1 failures. A higher level reconfiguration implementation (PU_R) has been added to redirect all the data that the failed PU was sending from its communication interfaces. PU_R monitors the performance of both PUs (PU_1, PU_2) and when it detects that any of them is down (time-based fault detection); it is reconfigured sending the data that was sending before through MVB and Ethernet. Consequently, Rt_1 is replaced by Rt_{12}: $T_1.ETH \rightarrow PU_R \rightarrow ETH \rightarrow PU_2$; and Rt_2 switches to Rt_{22}: $T_1.MVB \rightarrow PU_R \rightarrow MVB \rightarrow PU_2$.

All in all, the integration of the three scenarios in a single architecture results in a fault-tolerant architecture which copes with sensor, communication and PUs failures reusing already existing elements. Note that these scenarios have been tested isolated from the other functions comprising a real train, and hence, we do not have to deal with possible memory and bandwidth issues.

5. CONCLUSIONS & FUTURE GOALS

In massively networked scenarios there is room to optimize system architectures to improve system's dependability and reduce the overall cost. The proposed methodology provides the designer with indicators to support tradeoff design decisions between dependability and cost when deciding to optimize the use of system resources and allocation of system tasks on Processing Units (PUs). Generic Dependability Evaluation Model and Component Dynamic Fault Trees have been presented as a means to perform tradeoff analyses between dependability and cost, evaluating the influence of alternative redundancy and reconfiguration strategies.

Heterogeneous redundancies reuse system resources to provide compatible functionalities to a system function. Conversely, homogeneous redundancies add additional resources to replicate system functions. In the developed application example heterogeneous redundancies reduce the overall cost and improve system dependability with respect to the architecture without redundancies, while performing almost as well as homogeneous redundancies. Regarding the allocation of tasks on PUs,

the impact of centralized and distributed reconfigurations on system (un)reliability has been analyzed showing that the criticality of centralized reconfigurations is higher than distributed reconfigurations

For our future goals, we plan to integrate repair concepts and uncertainty analyses in the methodology to evaluate the influence of unknown (SW) failure rate values on system (un)availability. Moreover, we plan to address the automatic optimization of system architectures based on requirements specified as dependability and cost values to extract the best combination of homogeneous and/or heterogeneous redundancies. Lastly, the evaluation of the degradation of the functionality with heterogeneous redundancies may be addressed analyzing other factors than system failure probability.

References

[1] C.P. Shelton & P. Koopman. "*Improving System Dependability with Functional Alternatives*", In Proceedings of Dependable Systems and Networks 2004, pp. 295-304, (2004).

[2] J. Wysocki & R. Debouk. "*Methodology for Assessing Safety-Critical Systems*", International Journal of Modelling & Simulations, vol. 27, no. 2, pp. 99-106, (2007).

[3] R. Adler, D. Schneider, & M. Trapp. "*Engineering Dynamic Adaptation for Achieving Cost-Efficient Resilience in Software-Intensive Embedded Systems*", In Proceedings of Engineering of Complex Computer Systems, pp. 21-30, (2010).

[4] J. I. Aizpurua & E. Muxika. "*Functionality and Dependability Assurance in Massively Networked Scenarios*", In Proceedings of ESREL 2013, pp. 1763-1771, (2013).

[5] J. I. Aizpurua & E. Muxika. "*Model Based Design of Dependable Systems: Limitations and Evolution of Analysis and Verification Approaches*", International Journal on Advances in Security, vol. 6, pp. 12-31, (2013).

[6] B. Kaiser, P. Liggesmeyer, & O. Mäckel. "*A New Component Concept for Fault Trees*", In Proceedings of Safety Critical Systems & Software 2003, pp. 37- 46, (2003).

[7] H. Boudali, P. Crouzen, & M. Stoelinga. "*A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains*", In Proceedings of Automated Technology for Verification & Analysis, vol. 4762, pp. 441-456, (2007).

[8] G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, & N. Trapani. "*MatCarloRe: An Integrated Monte Carlo Simulink Tool for the Reliability Assessment of Dynamic Fault Tree*", Expert Systems with Applications, vol. 39, no. 12, pp. 10334-10342, (2012).

[9] G. Vinod, T. Santosh, R. Saraf, & A. Gosh. "*Integrating Safety Critical Software System in Probabilistic Safety Assessment*", Nuclear Eng. & Design, vol. 238, no. 9, pp. 2392-2399, (2008).

[10] SINTEF Industrial Management. "*Offshore Reliability Data Handbook*", (2009).

[11] IAEA-TECDOC478. "*Component Reliability Data for Use in Probabilistic Safety Assessment*", Technical Report, (1988).

[12] W. Wendai, J. Loman, & P. Vassiliou. "*Reliability Importance of Components in a Complex System*", In Proceedings of IEEE Reliability and Maintainability Symposium, pp. 6-11, (2004).

[13] International Electrotechnical Committee. "*Train Communication Network, IEC 61375*", (2007).

[14] J.I. Aizpurua & E. Muxika. "*Dependable Design: Trade-off Between the Homogeneity of Functions and Resources*", In Proceedings of DEPEND 2012, volume 1, pp. 13-17, (2012).