Dependable Design: Trade-Off Between the Homogeneity and Heterogeneity of Functions and Resources

Jose Ignacio Aizpurua, Eñaut Muxika
Department of Signal Theory and Communications
University of Mondragon
Spain
{*jiaizpurua,emuxika*}@*mondragon.edu*

***Abstract*—Designing a dependable system by reducing costs is a challenging issue. Traditional design strategies replicate resources in order to improve fault-tolerant capabilities of the system, which leads to increasing the hardware cost. Originated from over-dimensioning decisions, we outline an approach to identify and gather inherent compatible resources of the system to accomplish equivalent functions. Inference of reconfiguration strategies from the inherent design redundancies of the system not only decreases the hardware cost, but also maintains, or even improves the dependability of the system.**

***Keywords*-Dependable design, distributed and reconfigurable systems, shared (quasi) redundancy.**

I. Introduction

When designing a dependable system [1], redundancies improve system safety and availability. However, the aggregation of resources leads to higher costs and more failure sources. Consequently, reliability decreases. Therefore, a trade-off analysis between dependability and cost objectives is necessary to design a dependable system.

In order to perform a function within distributed networked control systems (NCSs), remote devices work in cooperation. A sensor performs a measurement function and sends it to a control algorithm through the network. The control algorithm acts in consequence and sends actuation commands to the remote actuator. Traditionally, sensors and actuators accomplish a single function, while processing units (PUs) handle multiple tasks. However, why not exploit sensor and actuator strengths to perform as many functions as they can?

Our design approach focus on NCSs operating under massively networked scenarios, where a lot of PUs and sensors are connected to a network for different purposes. E.g., a train where the resources of each car are replicated throughout the train cars or a building where room and floor resources are replicated.

In this paper, we propose a system design approach for the systematic identification of replaceable functions including those performed by sensors and actuators. To do so, the physical location is the key driver. Subsequent steps allow associating and deducing inherent design redundancies of the system. This approach allows improving specifically system availability and generally system dependability without additional hardware cost.

The remainder of this paper is organised as follows: Section II gives an overview of related research work. Section III describes the generic functional modelling approach. Section IV describes an overall reconfiguration process for designing a dependable system. Finally, Section V addresses limitations and future objectives of our research.

II. Background Research

This section is divided into two subsections: Subsection II-A points out inspirations of our research and Subsection II-B identifies conceptually aligned works.

A. Research Inspiration

A straightforward way to add redundancies to a system design is to explicitly replicate components. The objective of the added resources is to provide failover capabilities to a dedicated component failure. These replications are usually done using *passive* and *active redundancies*.

Shared-redundancy [2] and *quasi-redundancy* [3] concepts emerge from the utilization of components to compensate for a failure, despite not being primarily used with this objective. Replication of control functions over distributed processing units (PUs) is done in such a way, that failed functions are compensated using existing components. These redundancies are implemented reconfiguring the communication routes of the network and PUs.

Passive and *active redundancies* replicate the nominal operation of the failed function and *shared-* and *quasi-redundancies* make the failed function operate under degraded conditions. Note that *shared-* and *quasi-redundancies* are a form of *passive redundancies*, i.e., they work only when the primary resource fails.

In order to simplify the nomenclature, we name *homogeneous resources* those needed to perform *passive/active redundancies* where the nominal operation is replicated and *heterogeneous resources* those needed to perform *shared-/quasi-redundancies* where failure of the nominal operation is compensated (i.e., degraded operation).

We reuse existing concepts grouping them to consider the trade-off between the replication of resources and the reuse of existing ones. *Homogeneous resources* lead to increasing the hardware cost. However, the integration and implementation of these resources is not as difficult as with the *heterogeneous resources*. The identification of replaceable functions and the adaptation of the existing architecture to benefit from compatibilities are the main challenges. Therefore, our aim is to complement existing approaches with a method to identify *heterogeneous resources*. This process enables a systematic characterization of the replaceability properties of the system, including those involving sensors and actuators.

B. Related Works

There is an increasing interest in automotive, avionics and space industry for reusing existing hardware and/or extracting system reconfiguration behaviour.

DySCAS middleware [4] partially addresses our considerations using a context-aware adaptation mechanism, specified by execution and architecture aware contexts. The former context uses distributed policies to detect deviations and react, while the latter embeds meta-information of configuration reasoning (resource dependencies, QoS contracts, compatibility, composability and dependability) within dynamically reconfigurable components. The approach deals with task migrations to cope with hardware failures and network balancing. A global node dynamically maintains for the entire network the intentions of every node and decides the possible configurations based on their requirements. Each node locally performs admission control deciding if a task is schedulable considering resource limitations (memory, CPU, bandwidth) and optimization of resources. The middleware consolidates and disseminates the distributed information.

Adler et al. [5] proposed a component-based modelling and verification method for adaptive embedded systems. The approach aims at exploiting implicitly available redundancies to react to system failures. It provides methodological support for identifying and gathering reasonable system configurations. To do so, each port of the functional component is attached with a quality attribute (QA), which provides means to connect compatible components. Based on QAs, the adaptation behaviour of each component is determined with the required qualities for activation (preconditions) and influences on the provided qualities (postconditions). In order to ensure the causality of the reconfiguration sequences, well-definedness properties are verified by using model-checking and theorem proving techniques.

Integrated Modular Avionics (IMA) paradigm defines robust partitioning in onboard avionic systems so that one computing module (Line Replaceable Unit (LRU)) is able to execute one or more applications of different criticality levels independently. The standardized generic hardware modules forming a network leads to looser coupling between hardware and software applications.

SCARLETT project [6] aims at designing reconfigurable IMA architectures in order to mitigate the effect of failures implementing functional and mitigation functions. *Monitoring and fault detection* function aims at detecting component failures. Once a permanent failure is detected, the *reconfiguration supervisor* manages the modifications of configurations given the current configuration and failed module. *Verification activities* check the correctness of the system configuration and the loaded data in the LRU. The centralized supervisor determines a suitable configuration based on a reconfiguration graph, which contains all possible configurations. Reconfiguration policies and real-time and resource constraints, define the set of reachable safe transitions and states. In order to analyse the reconfiguration behaviour when failures occur, a safety model leads to finding the combinations of functional failures. Based on the same concepts, DIANA project [7] aims at distributing these functionalities. This approach improves availability of reconfiguration mechanisms at the expenses of relying on a complex, resource consuming communication protocol.

Based on the potential of the IMA paradigm as a means to provide fault containment strategies, Montano and McDermid [8] presented an autonomous dynamic reconfiguration system. Different information required for an effective dynamic reconfiguration (task scheduling, hardware resources, operating modes, mission objectives, faults and dependability requirements) is gathered based on interactive Constraint Satisfaction Problem theory. The approach divides hard constraints and soft constraints. While the former is compiled off-line the latter can be added and retracted dynamically. Additionally, human interaction is allowed by translating his requirements into soft constraints and weighting the reconfiguration constraints so that higher priority decisions can be controlled.

All these approaches address the integration of reconfigurability and dependability aspects. System reconfiguration requires being aware of the system health, its operating modes as well as the behavioural and dependability requirements. These tasks should be implemented in a (autonomous) centralized or distributed supervisor and coordinated by a communication algorithm. However, the systematic identification of compatible resources has received scant attention. Adler et al. [5] intuitively characterize the quality attributes of the system components so that inter-component compatibilities are identified. To the best of our knowledge, no one is identifying and gathering replaceable resources based on the physical location of hardware components. This viewpoint may provide a useful systematic characterization about the effect of placing components in determined places.

III. Generic Functional Design Model

The goal of this section is to formalize design concepts so as to provide a systematic consideration of system functions, resources and the relation between them. In order to identify systematically *compatible subfunctions*, this model accounts for the physical location early in the design phase. The system is characterized in a top-down manner, parting from a set of *high-level* (HL) functions (cf. Figure 1).

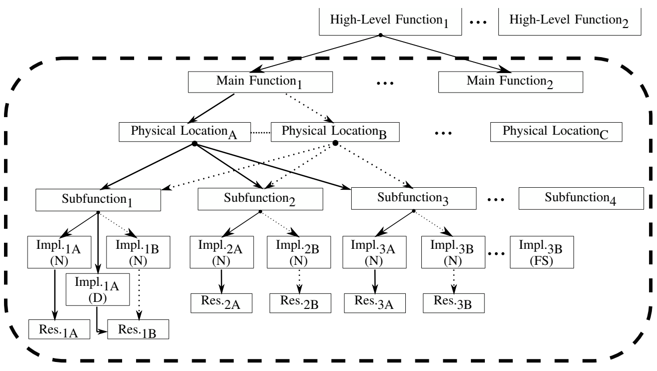

Figure 1. Generic Functional Design Model

Depending on the system HL functions, these are further refined into a set of *main control functions* (MFs). Our design considerations focus on system refinement from MFs downwards to limit the scope of the analysis without losing its generality. The *physical location* (PL) characterizes the place in which these MFs are performed. A single MF may cover different PLs or it may be replicated for each PL (e.g., Temperature Control). A set of *subfunctions* (SFs) define necessary and sufficient means to perform the MFs. Hence, the characterization of the system MFs is specified as follows:

MainFunction.PhysicalLocation.Subfunction.Implementation

There may exist different *versions* of SF implementations determined by the availability of resources. The resources provide means to perform a SF using a set of hardware resources, which may allocate software functions. Based on the system means to perform the same SF with different resources, we differ nominal, degraded and fail-safe versions.

The *nominal (N)* version, performs under initial functional design characteristics. The set of Input (I), Control (C) and Output (O) SF components necessary to perform the nominal MF, in conjunction with the necessary resources to address the system requirements, form the nominal design *configuration*.

When the I, C or O subfunction is lost due to the failure of some resource, the configuration to achieve an acceptable outcome may have to change. There may be subfunctions, which provide a *degraded (D)* but acceptable service, even in presence of faults. *Fail-safe (FS)* versions emerge from the need to cope with the insurmountable loss of resources, which result in hazard occurrences. Predetermined solutions should be defined so as to avoid these situations.

According to this classification, we define the concept of *compatible subfunctions*. Two SFs are compatible if their SFs match and they are within a compatible PL. This compatibility would determine the acceptable value for the produced outcome. The compatibility of the PL depends upon the examined SF component. The PL of each SF specify whether we are dealing with a zone-level (e.g., Train.car.zone) or specific-level (e.g., Train.car.zone.location) SF. For I/O SFs performed within the zone-level and depending on the I/O type itself, we consider that the difference (if it exists) between the produced outcomes of compatible subfunctions is acceptable (e.g., Temperature Control in adjacent compartments). However, specific PLs, confine the compatibility within a specific physical space. Generally, output SFs are gathered within this group, due to their specific actuation space. C subfunctions, do not have an explicit dependency on the PL. They are able to perform the C function provided it receives the corresponding I values of the specific PL.

Emerging from these concepts, we make a comparison between *homogeneous* and *heterogeneous resources* (cf. Table I). To do so, we centre on the nominal MF configuration (cfg) and those which use *homogeneous* and *heterogeneous resources*. Remember that since *heterogeneous resources* focus on reusing distributed compatible resources, the MF configuration will vary from the nominal design implementation.

Table I
Comparison between heterogeneous/homogeneous resources and nominal implementation

Resources	SF	PL	I	C	O	cfg
Homogeneous	=	=	=	=	=	=
Heterogenous	=	≡	≡,=	≡,=	≡,=	≡

same(=); compatible(≡)

IV. Reconfiguration Process

The process described throughout this section is based on the application of the generic functional design model (cf. Section III) to model, identify and gather compatibilities and extract customized reconfiguration mechanisms. We rely on a running example in order to discuss and evaluate the process.

A. Modelling Functions and Resources

The goal of the modelling process is to identify and gather compatible functions performed with alternative resources. The design model, makes these tasks possible, tracing from system functions towards physically distributed implementations. For instance, consider the temperature control (MF$_1$) within a train car (cf. Figure 2).

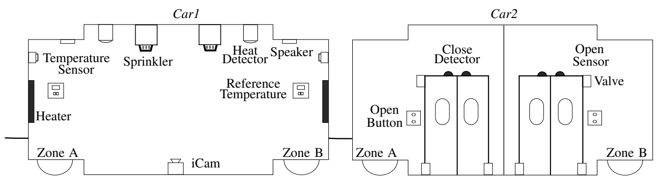

Figure 2. Train Physical Topology

From MF_1, temperature measurement, user's temperature reference, temperature control and heating SFs are characterized (cf. Table II).

Table II
CHARACTERIZATION OF TRAIN CAR TEMPERATURE CONTROL

Main Function	*PhysicalLocation*	*Subfunction*	*Implementation*
Temperature Control	Car1.ZoneA	MeasureTemp	$Sensor_A$
		RefTemp	$RefButton_A$
		TempControlAlgorithm	PID Control
		Heating	$Heater_A$
	Car1.ZoneB	MeasureTemp	$Sensor_B$
		RefTemp	$RefButton_B$
		TempControlAlgorithm	PID Control
		Heating	$Heater_B$

If we proceed to model all functions, *heterogeneous resources* are systematically identified. This process consist of matching SF and PL tokens, in order to identify compatible resources.

B. Identification of Possible Redundancies

The layered modelling of functions, resources and physical locations allows identifying inherent redundancies from multiple system functionalities. The systematic utilization of the described modelling process permits gathering initially different, but suitable, functions.

For instance, consider MF_1 for both compartments (zone A, B) of the train car (cf. Table II). Redundancies may appear at input and control SFs. Input redundancies originate from the utilization of a single temperature sensor for different zones. Similarly, contiguous compartment temperature reference values can be used to provide a degraded, but acceptable, subfunction. Control SFs may be implemented on any PU depending on its capabilities (e.g., memory, execution time).

Once *heterogeneous resources* and possible redundancies are identified, a strategy to avoid single point of failures (SPOFs) could be to add *homogeneous resources*. As an example, consider the management of *heterogeneous resources* at the actuator level. Unless actuators are supplied with explicit redundancies or compatible functions actuate in a shared physical space, *heterogeneous resources* become infeasible. Heating SF is not replaceable due to the lack of suitable SFs within the same space. In these cases, *homogeneous resources* should be supplied in order to provide switch over capabilities.

C. Resource Allocation

When analysing the resource allocation for MF_1, some architectural assumptions are made for exemplifying how to customize the approach to a fully distributed system. It is considered that sensors, user temperature references and actuators are distributed in such a way, that they need a PU in order to reach the rest of components using a network communication protocol.

Additionally, even if for the purposes of this example communication, fault detection and reconfiguration failures are not considered, we are expanding the modelling of functions and resources to include them. The idea we are developing is to attach fault detection algorithms to each subfunction so that component and communication failures can be detected. We also need to attach reconfiguration algorithms to each main function, which are responsible for changing the *homogeneus/heterogeneus resources* using the aforementioned fault detection outcomes.

Consequently, one sensor, one reference button, three PUs and a heater connected through a communication network constitute the nominal configuration for the train car temperature control for each zone (cf. Figure 3). Each PU allocates different control algorithms so as to assure functionalities in case of input subfunction failures: Open Loop (OL) algorithm manages the omission of temperature measurement and a default Set Point (SP_A) enables handling user's temperature reference failure. Each SF must be allocated to the available resources. To do this, the model defined in Subsection IV-A needs to be completed in order to address the resource allocation decisions:

$$
\begin{gathered}
MF_1.\mathrm{Car1.ZoneA.MeasureTemp.Sensor_A} \\
MF_1.\mathrm{Car1.ZoneA.MeasureTemp.Sensor_B} \\
MF_1.\mathrm{Car1.ZoneA.RefTemp.RefButton_A} \\
MF_1.\mathrm{Car1.ZoneA.RefTemp.RefButton_B} \\
MF_1.\mathrm{Car1.ZoneA.RefTemp.SP_A.PU_{A1,A2,A3}} \\
MF_1.\mathrm{Car1.ZoneA.TempControlAlgorithm.PID.PU_{A1,A2}} \\
MF_1.\mathrm{Car1.ZoneA.TempControlAlgorithm.OL.PU_{A3}} \\
MF_1.\mathrm{Car1.ZoneA.Heating.Heater_A}
\end{gathered}
$$

From the previous model, we can observe that there is only one implementation for the Heating SF and therefore, we can deduce that it is a SPOF. This method allows an straightforward identification of SPOFs.

D. Inference Process and Reconfiguration

Once resource allocation decisions have been adopted, this approach enables the extraction of alternative system configurations. For instance, for MF_1 in the train car, the system configurations transit from nominal (N) configurations, in which the initial resources are working (W) correctly, to degraded configurations. These configurations allow handling for example the failures of $sensor_A$ (D_a), both sensors (D_b), reference button$_A$ (D_c) and both reference button failures (D_d). Further degradation occurs when the communication network fails (D_e). These configurations are illustrated in the Table III. Note that the purpose of this example is not to provide an exhaustive analysis of all existing configurations, but rather we want to illustrate a subset of these configurations, to show the application of the method without losing its generality.

Table III
CONFIGURATION EXAMPLES FOR ZONE A

Compatible Tokens: Implementations	N	D_a	D_b	D_c	D_d	D_e
MF_1.Car1.ZoneA.MeasureTemp.Sensor$_A$	W	F	F	W	W	F
MF_1.Car1.ZoneA.MeasureTemp.Sensor$_B$		W	F			F
MF_1.Car1.ZoneA.RefTemp.RefButton$_A$	W	W	W	F	F	F
MF_1.Car1.ZoneA.RefTemp.RefButton$_B$				W	F	F
MF_1.Car1.ZoneA.RefTemp.SP$_A$.PU$_{A2}$					W	F
MF_1.Car1.ZoneA.RefTemp.SP$_A$.PU$_{A3}$						W
MF_1.Car1.ZoneA.TempControlAlg.PID.PU$_{A1}$	W	W		W	W	F
MF_1.Car1.ZoneA.TempControlAlg.OL.PU$_{A3}$			W			W
MF_1.Car1.ZoneA.Heating.Heater$_A$	W	W	W	W	W	W

We need to assign priorities to each implementation for the SF components. This allows an automated generation of configurations in the case of failure occurrences, e.g., the priorities for the MeasureTemp SF for MF_1.Car1.ZoneA would be {Sensor$_A$, Sensor$_B$}.

Figure 3 shows the network and the data flow between different PUs. The thick lines represent the physical measurements of the sensors. The solid lines represent the data transfers in nominal operation mode. The dashed lines represent data transfers in a degraded operation mode when PU_{A1} and PU_{B1} fail. Finally, the dotted lines with empty arrowheads represent the data transfers in a degraded operation mode when PU_{A1}, PU_{B1}, PU_{A2} and PU_{B2} fail. In this configuration, PU_{A3} and PU_{B3} manage the temperature using OL algorithms and default reference temperature values.

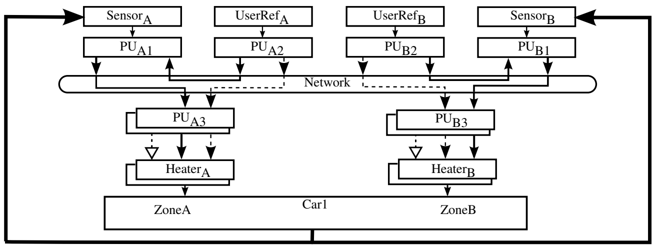

Figure 3. Train Car Logical Reconfiguration Topology

V. CONCLUSION AND FUTURE WORK

In this paper, a dependable design strategy has been sketched. Dedicated replication of system components satisfy the avoidance of SPOFs. However, in some environments it is feasible and desirable to make use of existing resources so that other compatible functions are supplied with *heterogeneous resources* (e.g., trains, buildings).

Identification and implementation of the proposed reconfiguration strategies will incur an extra cost. However, for a given dependability level, this approach would reduce the overall system hardware cost as *heterogeneous resources* are systematically identified. Note that this is only true in the previously mentioned environments.

We are developing the approach to support fault detection and reconfiguration. This will complete the method and we will be able to evaluate the dependability gains and cost trade-offs of this approach. Kazman et al. [9] propose a trade-off analysis method, which enable the architectural dependability-cost evaluation. This method could be integrated with this approach. The final goal is to obtain a complete method, where a quasi-optimal solution is obtained in a massively networked scenario.

In the cases where the dependability is critical, greater architectural details should be considered (e.g., power supplies, communication routes). This would allow evaluating the needed *homogeneus/heterogeneous resources*.

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, pp. 11–33, January 2004.

[2] J. Wysocki, R. Debouk, and K. Nouri, "Shared redundancy as a means of producing reliable mission critical systems," in *Proc. of RAMS'04*, 2004, pp. 376 – 381.

[3] J. Galdun, J. Ligus, J.-M. Thiriet, and J. Sarnovsky, "Reliability increasing through networked cascade control structure - consideration of quasi-redundant subsystems," in *IFAC Proc. Volumes*, vol. 17, 2008, pp. 6839–6844.

[4] R. Anthony, D. Chen, M. Törngren, D. Scholle, M. Sanfridson, A. Rettberg, T. Naseer, M. Persson, and L. Feng, *Autonomic Communication*. Springer, 2009, ch. Autonomic Middleware for Automotive Embedded Systems, pp. 169–210.

[5] R. Adler, I. Schaefer, M. Trapp, and A. Poetzsch-Heffter, "Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 20:1–20:39, Dec. 2010.

[6] P. Bieber, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, "Preliminary design of future reconfigurable ima platforms," *SIGBED Rev.*, vol. 6, no. 3, 2009.

[7] C. Engel, A. Roth, P. H. Schmitt, R. Coutinho, and T. Schoofs, "Enhanced dispatchability of aircrafts using multi-static configurations," in *Proc. of ERTS'10*, 2010.

[8] G. Montano and J. McDermid, "Autonomous and/or interactive constraints-based software reconfiguration for planetary rovers," in *Proc. of ASTRA'08*. ESA/ESTEC, 2008.

[9] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for Architecture Evaluation," SEI, Carnegie Mellon University, Tech. Report CMU/SEI-2000-TR-004, 2000.