Isolation of redundant and mixed-critical automotive applications: effects on the system architecture

1st Alessandro Frigerio
Eindhoven University of Technology
Eindhoven, The Netherlands
a.frigerio@tue.nl

2nd Bart Vermeulen
NXP Semiconductors
Eindhoven, The Netherlands

3rd Kees Goossens
Eindhoven University of Technology
Eindhoven, The Netherlands

***Abstract*—Future automotive systems, with Advanced Driving Assistance Systems and Autonomous Driving functionalities, will require fail-operational electronic systems. To achieve that, redundancy is a necessary technique, like in many other fields such as aviation. Moreover the applications have different safety requirements, from safety-critical related applications, for example for the driver replacement domain, to QoS-oriented applications, for example for the infotainment domain. Redundancy in mixed-criticality systems can be solved by physically separating system resources or by using isolated virtualized environments with e.g. hypervisors. There are costs associated to both solutions. In this work we describe a novel model we use to characterize a mixed-criticality automotive system and the analysis steps to obtain quantified metrics. The quantified metrics include cost, failure probability, total functional and communication loads, and total cable length, to compare the different solutions from a system-level perspective. We analyse the same set of mixed-criticality applications that represent a simplified automotive system in four scenarios. The architecture topology is either domain-based or zone-based, and we use either physical separation or virtualization to provide isolation. The obtained results show how the model and the analysis allows us to understand the trade-offs between the different solutions in specific applications scenarios, and how to vary the metrics used in the analysis to adapt to a different applications scenario.**

***Index Terms*—ADAS, ASIL decomposition, AV, functional safety, redundancy, safety-critical systems, virtualization**

I. Introduction

Functional safety is a key aspect for Autonomous Vehicles (AVs). The automotive electronic system must be reliable and fault-tolerant to always provide safe guidance to the vehicle, even in the presence of faults. The ISO26262 standard [1] describes the procedures to develop and verify the safety of an automotive system. Relevant for this work is the Automotive System Integrity Level (ASIL) decomposition technique which is described in the standard: a Functional Safety Requirement (FSR) can be decomposed into redundant FSRs with lower ASIL requirements if the redundant subsystems are independent. The combination of the resulting FSRs must be analysed at the system level, to make sure no Common Cause Faults (CCFs) can invalidate the independency requirement.

The main techniques used to provide independency, and so independence, are physical separation and resource virtualization [2]–[4]. With physical separation the redundant parts of an application are mapped to different hardware resources. These resources can be located in separate parts of the vehicle to avoid any type of common environmental failure condition such as electromagnetic interference. In the case of resource virtualization an additional software layer, called a hypervisor, is used to safely share hardware resources and to isolate application nodes. These two techniques are effective both for isolating application nodes in case of redundancy and in case of mixed-criticality applications.

In this work we analyse different architectural choices to understand which advantages and disadvantages the virtualization technique has. In particular our contributions are:

- The extension of an existing automotive system model and analysis flow to allow the description of virtualization with mixed-critical and redundant applications.
- The analysis of a simplified automotive system developed with four different architectures. The effects of virtualization are analysed in presence of mixed-critical and redundant applications. The parameters we use for comparing different solutions are the cost of the system, the failure probability of safety-critical applications, the total communication and computation loads, and the total cable length related to the resources required for the automotive system.

The rest of the paper is organised as follows: Section II describes the methodology we use to model and analyse an automotive system. Section III contains the example applications and architectures that we analyse. The results of the analysis are explained in Section IV. Finally, we analyse the related work in Section V and conclude in Section VI.

II. Methodology

A. Automotive Electronic System Model

In this work we compare different architecture solutions with quantifiable metrics. We adopt and extend the model from [5], that divides the automotive system into three layers: *application*, *resource*, and *physical*. Each layer describes the characteristics of a different part of the system: the *application* layer contains the application nodes, their requirements (ASIL, functional or communication loads), and the logical connections between them; the *resource* layer is formed by the hardware resources, their specifications (ASIL, maximum functional and communication loads), and their physical connections; the *physical* layer contains the locations in which the hardware resources can be positioned and their spatial coordinates. Each layer is described by a graph, $G_a = (V_a, E_a)$,

$G_r = (V_r, E_r)$, $G_p = (V_p, E_p)$. In addition two sets of edges define the mapping of the application nodes to the hardware resources and the mapping of the hardware resources to the physical locations, $E_{ar} : V_a \rightarrow V_r$ and $E_{rp} : V_r \rightarrow V_p$.

As an extension to the existing model, when a hypervisor is used each Virtual Execution Platform (VEP) will be present in the resource graph as a separate resource, with a link to the original resource used, and application nodes can be mapped to them. The ASIL specification of each VEP depends on the original resource's and the hypervisor's specifications.

We analyse trade-offs between different implementations based on five metrics: the cost of the system, the failure probability of the safety-critical applications, the total communication and functional loads, the total cable length. In Section II-B we describe how to characterize redundancy in the model and in Section II-C we show how we can calculate these parameters on the modelled system. When analysing physical separation against virtualization either solution can be best, depending on the differences between the two implementations and to the costs associated to the two solutions.

B. Characterizing Redundancy

Redundancy is used in automotive and in safety-critical systems to provide fault-tolerant and fail-operational systems. As mentioned in Section I, redundant functionalities must be independent, and either physical separation or virtualization is applied. As in [5], we identify redundancy in the modelled systems with a pattern. When part of an application is redundant, the input data that comes from a non-redundant part must be forwarded to the redundant branches. The node that performs this operation is called *splitter*. At the end of the redundant branch, a node called *merger* must decide which of the different redundant outputs to forward to the following part. For example a splitter can be implemented by broadcasting the message to multiple nodes, and the merger can be a N-out-of-M voter. These functionalities exist both in the application layer and in the resource layer. Note that these operations are single points of failure, thus have high safety-critical requirements.

Figure 1a shows an example of physical separation for the redundant system that performs the processing operations *proc1* and *proc2*. The redundant parts of the application are mapped to different hardware resources, which are then connected to a splitter and a merger via separate communication resources. The CCFs are related only to the resource(s) on which the splitter and merger operations are performed.

In case of virtualization, as shown in Figure 1b, we map the redundant nodes to separate VEPs created on the same hardware resource *ECU*. The CCFs are related not only to the splitter and merger, but also to the original resource that is used, its location, and the hypervisor used. In this example a single input communication resource and a single output communication resource are shared. They are now also a CCF source and must be reliable enough for the redundant communication. Compared to the physical separation scenario, fewer communication resources are used, reducing the total

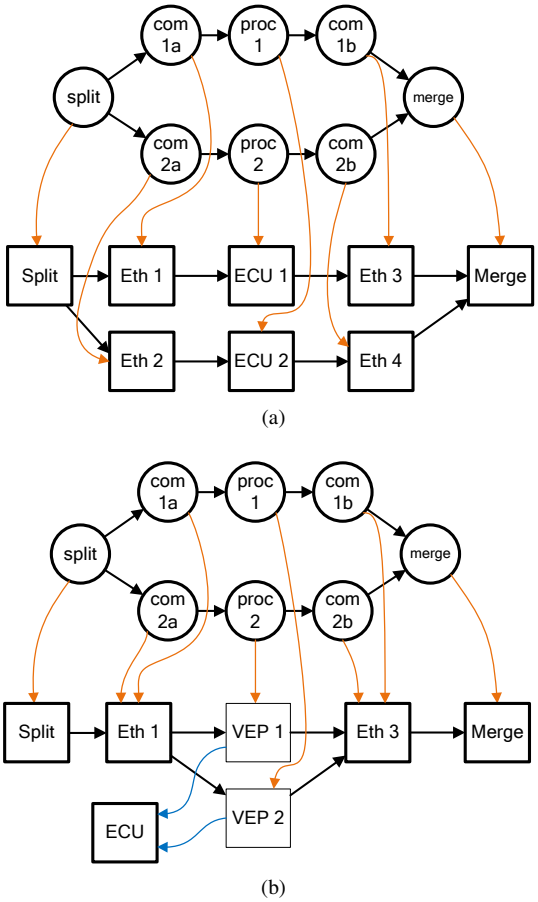

Fig. 1. Phyiscal separation (a) and virtualization of resource ECU (b) for independence of redundant application nodes. Black arrows represent same-layer logical interconnections, orange arrows are inter-layer mapping edges, blue arrows link the VEPs to the original resource in the resource layer.

wiring harness of the system. The bandwidth requirements for the single communication resources are however higher, since the communication is condensed on the two resources instead of being divided to four resources.

We identify three scenarios for redundancy in virtualized systems related to the position of the splitter and merger nodes:

1) Splitter and merger operations in separate VEPs. This scenario would allow inter-VEP communication, creating a possible break of the isolation that the hypervisor provides;
2) Splitter and merger operations as hypervisor functionality. This requires the hypervisor software to provide the tools to perform the splitter/merger operations or to be user-accessible;
3) Splitter and merger operations performed in separate hardware resources. This scenario, shown in Figure 1b, does not require any modification of the hypervisor functionality.

For this work, we will use only the third scenario, since it reuses commercial off-the-shelf hypervisors as is.

C. Calculating the comparison parameters

Once a system is described in the model, we calculate five parameters that we use to compare different solutions:

1) The first parameter is the *cost* of each application. We base this cost purely on the ASIL specification of the utilized

resources according to Table I. In case a hypervisor is used, an additional cost is added to the resource that is being virtualized based on the ASIL specification of the hypervisor itself and of the provided VEPs according to Table II. This cost reflects the additional complexity introduced by the use of an extra software layer and the hypervisor.

TABLE I
RESOURCES COST METRIC (ARB. UNIT) [6].

Resource Type	QM	A	B	C	D
Functional	5	50	500	5000	50000
Communication	4	40	400	4000	40000
Sensor / Actuator	8	80	800	8000	80000
Splitter / Merger	1	10	100	1000	10000

2) The second parameter is the *failure probability* of each application, in particular the safety-critical ones. A fault tree is generated from the modelled system to calculate this. This step is based on [7], where the application graph is traversed from actuators to sensors and at each visited node a pattern is added in the fault tree. Figure 2 shows the events and base events that are added to the fault tree for the application node *n*. In case of a *merger* node the *Input Failure* event would be connected to an *AND* gate instead of an *OR* gate. In case of a non-virtualized resource, the possible failures are related to the software nodes mapped on the same resource, to the resource itself, or to the physical location on which the resource is positioned. Note that we assume that nodes mapped on a functional resource are not *fail-silent* with respect to the other nodes that share the functional resource, unless they are isolated in a VEP. Nodes in the same VEP are also not *fail-silent* with respect to each other. In case of virtualization, the hypervisor is an additional cause of failure, shared by all the nodes mapped on that particular resource. In case of communication resources, we assume that the communication physical layer and protocol provide isolation between data from different sources, and previously transmitted data cannot affect the current transmission.

A failure rate λ is assigned to each base event of the fault tree. Table III shows the failure rates that we use in this work for the base events, based on [7], with the addition of hypervisor and software failure. The location failure rate does not depend on the ASIL value, and we consider a value of 10^{-11} failures per hour. We separate the failure events related to the failure of the splitter or merger resources and of the hypervisor. We assume a lower failure probability for these events because of their safety-oriented nature.

TABLE II
HYPERVISOR COST METRIC (ARB. UNIT).

		VEPs				
		QM	A	B	C	D
Hypervisor	QM	1	-	-	-	-
	A	10	100	-	-	-
	B	10	100	100	-	-
	C	100	100	100	1000	-
	D	100	100	100	1000	10000

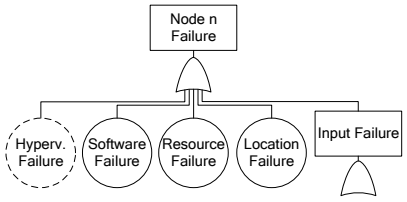

Fig. 2. Fault tree pattern generated from node *n*.

TABLE III
BASE EVENTS FAILURE RATES (FAILURES/HOUR).

Base Event Type	QM	A	B	C	D
Location	10e-11				
Splitter/Merger/Hypervisor	10e-6	10e-7	10e-8	10e-9	10e-10
Other base event	10e-5	10e-6	10e-7	10e-8	10e-9

The software failure rate is related to the ASIL value of the least safety-critical application node mapped on a specific functional resource or VEP. This corresponds to our assumption of non fail-silent nodes when mapped on the same resource or VEP.

3–4) The next parameters are the *functional* and the *communication loads* of an application. Each application node has a functional or communication load property. In our examples, we assign to the load values of each node a representative number, proportional to realistic computational or communication effort required to process each part of the application. The total loads of an application are calculated in each different implementation.

5) The last parameter is the *total cable length*. The Manhattan distance between the hardware resources is calculated based on the physical space coordinates assigned to the third layer. For each communication resource, the distance between the two resources that it connects (or the maximum distance in case of buses) is considered as part of the total cable length. In this work we consider normalized 2D coordinates ranging from -1.0 to 1.0 in both directions.

III. EXAMPLE APPLICATION

A. Software Applications

The applications in an automotive system are generally mixed-critical. For the analysis purpose we use a simplified automotive system, where three applications are present. Figure 3 shows the graphs that represent these three applications, with the communication or functional load annotated to each node. For clarity purpose, we identify three domains in the example automotive system: the *driver replacement* domain, which is safety-critical and has Advanced Driving Assistance Systems (ADAS) and AVs characteristics, and the *body and comfort* and *infotainment* domains with QoS characteristics. More complete descriptions of the automotive domains can be found in [8] and [9]. The first application we analyse is safety-critical (ASIL D), it collects environmental information with multiple sensors, analysing the scenario and providing trajectory and speed control data for the actuators. The second and third applications are part of the body and comfort and the infotainment domains respectively. The body and comfort application manages the in-vehicle temperature, while the

infotainment one is a surround view application that provides the driver a 360 degrees view of the vehicle's surroundings. Both of them are QoS applications (QM).

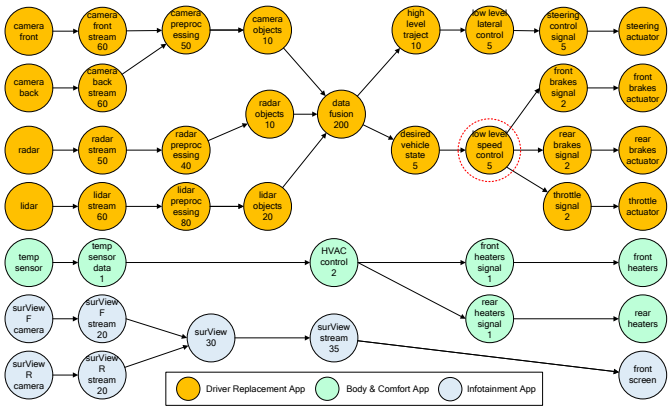

Fig. 3. The three example applications analysed in this work. Each node is annotated with its functional or communication load. The *low level speed control node* marked in red is made redundant in the experiments.

Each application has different requirements on the system: the safety-critical one requires high bandwidth on the sensing part, low bandwidth in the actuation part, and high computational loads to process the data. The body and comfort application has very low load requirements, while the infotainment has moderate load requirements.

For readability, redundancy is not shown in Figure 3. We introduce redundancy in the *low level speed control* node of the driver replacement domain application, marked in red, and it will have a pattern similar to the one shown in Figure 1 for the node *proc*. The resource on which it is mapped is redundant as well. Depending on the architecture topology used, which will be discussed in the next section, this resource can either be the domain-controller (for domain-based architectures) or the central unit (for zone-based architectures). The application nodes that share the resource will be redundant as well.

B. Hardware Resources

The hardware resources depend on the architecture topology choice. In this work we consider two options:

1) *Domain-based* architecture, in which each sensor and actuator is connected to the respective domain controller, which is then connected via the backbone network to the central unit;
2) *Zone-based* architecture, in which each sensor and actuator is connected to the closest zone controller based only on its proximity. Each zone controller is then connected via the backbone network to the central unit.

In both cases we consider a controller-based approach, which means that the controllers have some computational power and the application nodes are processed by them when possible. In the zone-based topology we assume that an application node can only be processed by the central unit when it requires inputs or sends its outputs from/to multiple zones. This scenario cannot happen in the domain-based topology since each sensor and actuator related to a domain are connected to the same controller. The driver replacement application node *data fusion* must be processed centrally due to its high computational load requirements.

Figure 4 shows our zone-based hardware resources implementation, in which we divide the system into two zones (front and back) plus a star backbone network and a central unit. Mixed criticality on the zone controllers exists, compared to the domain-based topology in which the domain controllers only have application nodes belonging to the same domain and so with the same ASIL requirements.

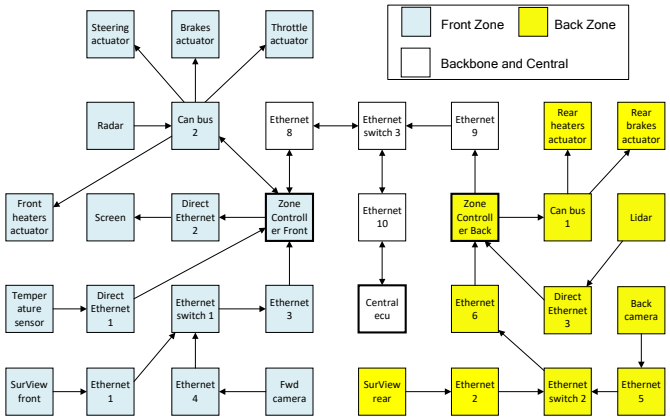

Fig. 4. Zone-based implementation of the hardware resources.

As mentioned in Section I, we use the physical isolation or the virtualization techniques to either isolate redundant nodes or mixed-critical nodes. No mixed criticality is present in case of a **domain-based topology**, so the isolation is only required by the redundant nodes of the application (the redundant low level speed control nodes):

- **Physical separation** (Dom Phys): the nodes that require isolation are mapped on the physically separate redundant driver-replacement domain controllers, which have lower ASIL specification due to the ASIL decomposition technique. All the other nodes that are processed on the domain controllers must be redundant as well, so that ASIL decomposition can be applied. An alternative would have been to add a third domain controller with safety-critical characteristics to process the other nodes, which would increase the total cost of the system resources.
- **Virtualization** (Dom Virt): the redundant low level speed control nodes are mapped on separate VEPs with lower ASIL specification on a single driver-replacement domain controller running a hypervisor. We add a third safety-critical VEP so that we map the other non-redundant nodes to that. We will obtain lower communication and functional loads by limiting the amount of redundancy in the application, and compared to the physical separation solution no extra cost is added to the system for this.

In case of the **zone-based topology** nodes with different ASIL requirements should be mapped to the zone controllers or to the central unit based on the belonging of their sources and outputs to one or multiple zones.

• **Physical separation** (Zone Phys): multiple zone controllers for both the zones are required because of mixed-criticality. The same applies for the central unit. Moreover the redundant low level speed control node must be mapped on physically separate redundant central units, for a total of three different central units.
• **Virtualization** (Zone Virt): VEPs are used for both mixed-criticality and redundancy. Three different VEPs are used in the single central unit, and two VEPs are used in each zone controller, since the applications have two criticality levels (ASIL D and QM).

IV. Experiments

In this section we calculate the comparison parameters on the example system described in Section III. We analyse the system in four different scenarios: domain-based with physical separation, domain-based with virtualization, zone-based with physical separation, and zone-based with virtualization.

The physical separation and the virtualization techniques are used for two reasons: we require isolation to avoid interferences in case of mixed-criticality application on the same functional resource, and we also require isolation in case an application has redundant parts.

In domain-based scenarios, no mixed criticality is present in the system, as every domain is confined in its own subsystem. The application functional nodes, with the exception of the data fusion node which requires high computational capabilities, are mapped to the local domain controllers, which for the driver replacement domain are made redundant to allow the mapping of the redundant low-level speed control node. All the other application functional nodes mapped on the driver replacement domain controller will be redundant as well.

In zone-based scenarios, the zone-controllers and the central unit will process application nodes belonging to multiple domains, and so with different safety requirements. In these scenarios the larger part of the safety-critical application will be executed in the redundant central units, since the nodes require information from sensors belonging to different zones.

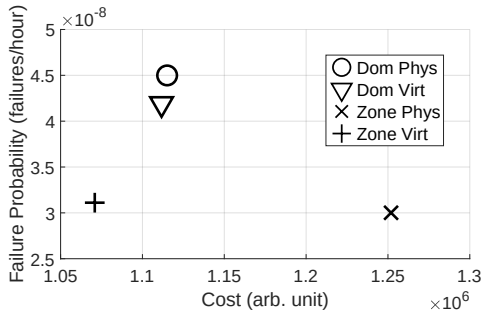

Fig. 5. Cost vs Failure Probability.

Figure 5 shows the values of the first two parameters, total cost of the system and failure probability of the safety-critical application, calculated on the modelled system by using the metrics described in Section II-C and the generated fault tree. The zone-based topology with physical separation is the most expensive, since all the controllers and the central unit must be duplicated for both redundancy and mixed-criticality, which in the virtualized solution become VEPs on the same resources.

When using safety-oriented resources to perform the splitter and the merger operations, their failure probability will be lower than a single-point of failure solution. This explains the reason why the failure rate of the domain-based topology is higher than the zone-based: in domain-based topologies the driver-replacement application node *data fusion* is processed by the non-redundant central unit, while in the zone-based topology the central unit is redundant, and the dominant failure rates are the ones of the safety-critical splitter and merger.

We observe opposite trends when using virtualization: in the domain-based topology the failure probability decreases, while in the zone-based it increases. This result depends on the application mapping and hardware configuration. For example in the domain-based scenario it was possible to share more communication resources as seen in Figure 1b, reducing the system complexity and the number of possible faults due to resources failures.

Figure 6 shows the communication and functional loads of the system based on the values given to each application node. The zone-based topologies, despite their lower failure probabilities, suffer in terms of total communication and functional loads: more raw data (with high bandwidth) from the sensors is transmitted towards the redundant central units, which are also performing the most computational heavy task redundantly. Virtualization does not help the zone-based

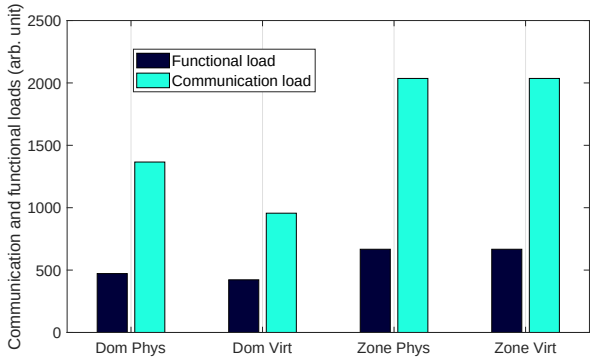

Fig. 6. Communication and Functional loads.

topology in reducing these two parameters.

In the domain-based scenario the sensor raw data stops at the domain controllers to be processed, resulting in a lower utilization of the backbone network. Moreover, thanks to virtualization the preprocessing part of the application is not redundant, cutting in half the associated communication and functional loads.

Figure 7 shows the total cable length based on the normalized 2D coordinates associated to the physical layer. Another advantage of zone-based topologies is a lower total cable length. Moreover, virtualization removes the need to duplicate communication links, thereby requiring fewer links, and hence a lower total cable length.

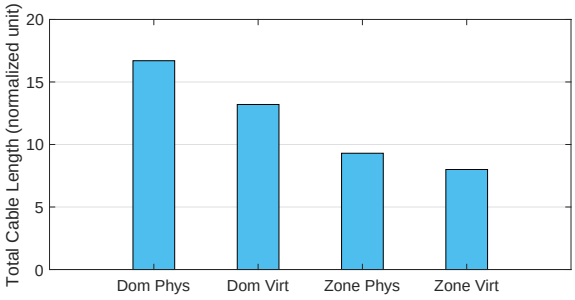

Fig. 7. Total cable length.

We understand from these experiments that virtualization does not always reduce the analysed parameters in an automotive system. Note that these results are obtained with the cost and failure rates metrics described in Section II-C and are calculated in specific architecture scenarios. For example, the failure probability of the system in domain-based topologies when virtualization is applied is reduced, despite introducing an additional CCF cause due to the possible failure of the hypervisor. If the failure rate λ_{hyp} would be more critical, the result could be different, and for example exceed the maximum failure probability that an ASIL D system can have, since the hypervisor's failure rate directly impacts the system failure probability. However our method is general and with the analysis that we propose these metrics can be adapted to a particular solution and the new parameter can be calculated to be compared with a different implementation.

V. Related Work

The literature related to Advanced Driving Assistance Systems and Autonomous Driving touches many different fields. In this work we focus on system-level descriptions of the system and on functional safety. The ISO26262 standard [1] provides guidelines to develop a safe automotive system, focusing on random and systematic faults events, and ensuring the functionality of the system in case of a system failure. It describes the ASIL decomposition technique which we use to decompose FSRs into redundant ones, which can be assigned to redundant parts of the system. In this paper we follow these guidelines and we describe methods to provide the isolation required by the ASIL decomposition technique to validate it.

To describe the automotive system, we extend the model of [5], where the authors describe a three-layer model to characterize an automotive system and define the *splitter* and the *merger* functionalities to identify redundancy patterns. A splitter is a node that replicates data on its input port to its outputs, connected to redundant parts of the system. A merger decides which of its inputs, connected to the redundant parts, should be forwarded to the following nodes. We extend their methodology to add the capability to describe and analyse virtualization and mixed-criticality.

The use of virtualization for automotive systems has been discussed in different works. For example the authors of [10] describes some advantages in terms of isolation due to using the VirtIO hypervisor, while the authors of [11] describe the implementation of the VOSYSmonitor hypervisor. The authors of [12] analyse low-level implications of virtualizing automotive control units. In our work we to take a step back to understand the implications that the use of virtualization has on the automotive architecture, in terms of higher level parameters such as the total cost, the failure probability of the applications, the total communication and functional loads, and the total cable length of the system.

VI. Conclusion

In this work we describe a methodology to analyse an automotive electronic system from a system-level point of view. An existing model is used to characterize the system, from which a different fault tree graph for each application is generated. We extend the model to allow the description of virtualization, a technique that is used to provide the isolation required for redundant and mixed-critical applications. We calculate five parameters, the total cost, the failure probability of the applications, the functional and communication loads, and the total cable length, to quantify and compare different solutions. As an example, we analyse a system composed by three applications in four different scenarios, based on the architecture topology and the use of virtualization. The numerical results are based on the characterization of the specific applications and hardware architectures we use, but can be adapted to any other automotive electronic system. We show how virtualization can provide improvements over physical separation in some particular situations, and how to quantify these advantages.

Acknowledgment

The work in this paper is supported by the TU/e Impuls program, a strategic cooperation between NXP Semiconductors and Eindhoven University of Technology. This research was supported through PENTA project HIPER 181004.

References

[1] ISO 26262-2018: Road vehicles - functional safety. 2018.
[2] Reinhardt, D., et al.: An embedded hypervisor for safety-relevant automotive E/E-systems. In: SIES, 2014
[3] Saidi, S., et al.: Future automotive systems design: research challenges and opportunities. In: CODES, 2018
[4] Aeronautical Radio Inc.: Avionics application software standard interface. ARINC Specifications 653, 2005
[5] Frigerio, A., et al.: A generic method for a bottom-up ASIL decomposition. In: SAFECOMP, 2018
[6] Murashkin, A., et al.: Automated decomposition and allocation of automotive safety integrity levels using exact solvers. In: SAE Int. J. Passeng. Cars - Electron. Electr. Syst., 2015
[7] Frigerio, A., et al.: Component-level asil decomposition for automotive architectures. In: SSIV, 2019
[8] Stolz, W., et al.: Domain control units - the solution for future E/E architectures? In: SAE Technical Paper, 2010
[9] Reger, L.: The EE architecture for autonomous driving. A domain-based approach. In: ATZ Elektron Worldw, 2017
[10] Lampka, K., et al.: Using hypervisor technology for safe and secure deployment of high-performance multicore platforms in future vehicles. In: ICECS, 2019
[11] Lucas, P., et al.: VOSYSmonitor, a TrustZone-based hypervisor for ISO 26262 mixed-critical system. In: FRUCT, 2018
[12] Rajan, A., et al.: Hypervisor for consolidating real-time automotive control units: Its procedure, implications and hidden pitfalls. In: JSA, 2018