Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments

Project Acronym:

EMC²

Grant agreement no: 621429

Deliverable no. and title	**D7.2 – Final automotive use cases implementation and evaluation**	
Workpackage	WP7	Living Lab Automotive
Task / Use Case	T7.1-T7.6	All tasks in WP7
Subtasks involved	All subtasks in all tasks in WP7	
Lead contractor	Infineon Technologies AG Werner Weber, werner.weber@infineon.com	
Deliverable responsible	VOLVO Thomas Söderqvist, thomas.soderqvist@volvo.com	
Version number	v1.0	
Date	2017-03-30	
Status	Final	
Dissemination level	Public (PU)	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
16F	VOLVO	Thomas Söderqvist	1, 2.1, 2.7, Appendix 6
17I	Technolution	Dave Marples	2.2, Appendix 1
15M	IXION	Nura Garcia	2.3, Appendix 2
02A	AVL	Georg Macher	2.4, Appendix 3
11B	CRF	Alberto Melzi	2.5, Appendix 4
13B	CSOFT	Joao Rodrigues	2.6, Appendix 5
02F	ViF	Florian Pölzlbauer	2.7, Appendix 6
16L	ARCCORE	Karl Erlandsson	2.7, Appendix 6
16A	Chalmers	Ioannis Sourdis, Stavros Tzilis	Appendix 6

Document History

Version	Date	Author name	Reason
1.0	2017-03-28	See authors table above.	Final version
	2017-03-30	Alfred Hoess	Final editing and formatting; deliverable submission

Executive Summary

This document is the Deliverable **D7.2 – Final automotive use cases implementation and evaluation** of the EMC^2 project. The primary objective of D7.2 is to present the third and last iteration of the implemention and evaluation of the WP7 Automotive Applications six use cases. The scope of the D7.2 is set based on the innovation cycles and the milestones as defined in the EMC^2 project. This deliverable describes activities performed to address the business needs, Key Performance Indicators (KPIs) and high-level requirements formulated in the D7.1 deliverable.

Table of contents

List of figures

List of tables

1. Introduction

1.1 Objective and scope of the document

This document gives for each use case a description of the final implementation and the evaluation results. The design of the use cases and preliminary evaluation results are previously described in the deliverables D7.3, D7.5, D7.7, D7.9, D7.11 and D7.13, one per use case. The following use cases and lead partners are covered:

Table 1: Automotive use cases and partners

Use case	**Lead partner**	**Participants as listed in the TA**
ADAS and C2X	TECHNO	BMW, DENSO, EB, NXP-GE, IFAT, TTT, HIB, NXP-NL, TUE, TNO, TOMTOM
Highly automated driving	IXION	HUA, HIB, UoMAN, CHALMERS
Design and validation of next generation hybrid powertrain / E-Drive	AVL	IESE, IFAT, AIT, IFAG, EB, SFR, TUW, TNO, BUT, Tecnalia, VIF
Modelling and functional safety analysis of an architecture for ACC system	CRF	DITEN
Infotainment and eCall Application Multi-Critical Application	CSOFT	AICAS, CISTER, HIB, INESC-ID
Next generation electronic architecture for commercial vehicles	VOLVO	KTH, Chalmers, ArcCore, ViF, TTTech, SICS, Magillem, Systemite, ALTEN, IFAT

1.2 Structure of the deliverable report

The main body of this document contains a common introduction for the WP7 Automotive Applications. Next part contains for each use case an abstract and a summary of the evaluation.

More details, like use case subtask descriptions, high level requirements, KPIs and detailed evaluation results per use case are described in one appendix per use case.

The intention with this structure is to keep the main body of this document limited in size and the reader can get a summary of all use cases by reading just this main part. Further details about the use cases are given by reading the appendices.

2. WP7 Automotive Use Cases

2.1 Introduction

The driving sources for the business and technical needs in WP7 are the automotive use cases within the different technical areas. The automotive use cases will leverage the technological advances developed in the WPs and particularly contribute to showcasing innovations in the following areas:

- Novel ADAS following large-scale software integration and full exploitation of available resources.
- Development, deployment and practical evaluation of Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) applications.
- EMC^2 Electronics Power Management: Requirement collection, concept development and HW demonstration.
- Master the next challenges on the road to fully automated driving by optimizing the use of HW/SW resources of the proposed multi-core embedded cloud and exploiting them in a real-life autonomous driving scenario.
- Next-generation hybrid powertrain control that are built following model-based engineering and new programming paradigms, and are supported by simulation.
- Develop and showcase new architectural solutions for electric and hybrid vehicles, in particular researching and developing contactless charging solutions.
- Exploitation of a methodology for supporting the application of ISO 26262 standard, reducing time of system analysis and design, increasing at the same time design robustness from the functional safety point of view.
- Demonstrator development realizing and evaluating an integration of mixed criticality applications using mainly COTS.
- Harvest the potential of multicore CPUs for mixed-critical applications in the next generation of Electrical and Electronic (E/E) architecture for commercial vehicles.

For information about the relation between the listed technology areas and the workpackages WP1 – WP6 see the technology cockpit chart.

2.2 T7.1 UC ADAS and C2X

2.2.1 Abstract

The objective of Use Case 7.1 was to investigate and implement a system architecture that combines the potential of existing technologies emerging for the ADAS and next generation vehicle infrastructure markets with new communication and EMC^2 architectures to deliver strategies for addressing these markets using tested and field qualified approaches.

Use Case 7.1 is focused on Objective 36 of the originally stated, project level goals and objectives:

- Implement and integrate several applications (including safety-critical applications) on a single MCMC platform on a COTS system – *measured through the number of control units in the vehicle.*
- Determine the added value of such a platform for these applications, by integrating and expanding their functionality and by anticipating their longer-term opportunities as devised in the WP's – *measured through the efficiency of control algorithms.*
- Assess the impact of interfacing external systems (in particular C2X communication) on this platform, e.g. in terms of reliability, latency and security – *measured through the degree of integration of applications of multiple criticality levels.*

The final output from the living lab is a set of demonstrators of the activities of the project, covering deliveries both built specifically for the demonstrator and as technical elements from the individual work packages and then migrated into WP7.1. The component parts of this presentation are held together by the common theme of making ADAS and C2X functionality more functional, more reliable, cheaper and more performant via the application of EMC^2.

As previously noted in D7.4, the lab has been structured into a number of 'Work Streams' that have resulted in a number of developments that have been implemented by smaller groups of companies both within WP7.1 and across the technical work packages of EMC^2.

During this final year work has been carried out on integrating these individual components into larger systems that fully demonstrate the ADAS objectives intended by the project. There are two distinct 'centres of gravity' for this integration process;

- A **Lab Based** demonstrator focussed on the Technolution Model Vehicle.
- A **Field Based** demonstrator focussed on the TNO Vehicle.

The concept of this integration process and the Basis Demonstrators that result from it were documented in D7.4. Within this document we present an overview of the evaluation results and key learnings per partner. It is the nature of a Living Lab that the journey is at least as important as the destination and materials provide information on key activities that have been conducted by the individual partners.

2.2.2 Evaluation results

In terms of the original Obective 36 activities,

● Implement and integrate several applications (including safety-critical applications) on a single MCMC platform on a COTS system: Demonstrated on the platforms of Technolution, TomTom, Denso, NXP, TUE and TNO with considerations of fault tolerance, reliability, partitioning for design simplification and power optimisation all being taken into account.

● Determine the added value of such a platform for these applications, by integrating and expanding their functionality and by anticipating their longer-term opportunities as devised in the WP's: The added value of MCMC platforms has been extensively demonstrated within this use case. ADAS demands high levels of availability from its component systems and it has been shown how this availability level can be met via various combinations of system partitioning, multi-redundancy and communicating sub-systems.

● Assess the impact of interfacing external systems (in particular C2X communication) on this platform, e.g. in terms of reliability, latency and security: Investigated via the integration of the NXP-DE radio, the NXP and Technolution 802.11p & Ultra Wideband Radios and the TomTom connected platform.

The formal evaluation is carried out in consideration of the WP7.1 business needs that were presented in deliverable D7.1, Section 2.3. In the interest of brevity the reader is directed to that document for a detailed description of the Business need and cross reference to the High and Low Level requirements;

Table 2: Business needs for use case 7.1

No	Title	KPI	Result
BN_T7.1_01	Testing and Validation	*Reduced Testing and Validation:* Reduce the number of hours required to test an ADAS system and the number of different vehicles and situations for which the ADAS system needs to be put to the test.	**Pass:** Observed reduction in testing and validation through separation of concerns. Reduction in product complexity due to partitioning and reduction in n^2.
BN_T7.1_02	Product Offer	*Enhanced Marketability:* The ability to sell ADAS systems in the market and to maintain and extend them in a cost effective and	**Pass:** New features (e.g. dynamically loadable functionality, new navigation capabilities) lead directly to enhanced marketability.

		performant manner shall be enhanced.	
BN_T7.1_03	Real Time Response	*Increased penetration of EMC^2 Systems:* EMC^2 multi-core systems are deployed in commercial systems and deliver benefits across defined axes including power consumption, maintainability and real-time response.	**Not Determined, but likely Pass:** It is extremely likely that EMC^2 systems will be widely deployed in future due to their addressing of BN_T7.1_02 and thus increased demand, however the time lag to commercial deployment means this has not been observed to date and commercial sensitivities preclude partners from disclosing their advanced plans.
BN_T7.1_04	Mixed Criticality Software Stacks	*Reduction of distinct in-vehicle platforms:* The number of distinct and differentiated platforms in the vehicle is reduced as functionality is combined onto single physical entities by means of EMC^2 functionality, partitioned only on software development concerns rather than performance concerns.	**Pass:** Even in the context of the developments for the living lab the number of distinct ECUs has been seen to reduce (e.g. the first generation control platform for the Technolution Vehicle had an ECU per wheel which were, in the second generation, subsumed into the EMC^2 low level processor.)
BN_T7.1_05	Safety, Privacy and Security	*Maintenance of systemic invariants:* Third party audit of ADAS systems from safety, security and privacy perspectives is successful while the systems still deliver their intended benefits.	**Pass:** The partitioning of the system while maintaining a homogeneous development platform is observed to provide stronger invariants than a multi-ECU disparate development environment where homogeneity of requirements is only enforced by document exchange.
BN_T7.1_06	Legacy Migration to Multi-core	*Retention of legacy investment:* Substantial proportions of time-served, mature code are retained despite the migration to an EMC^2 compliant system.	**Pass:** Observed very obviously in the DENSO use case which re-uses existing ECU code in a new environment with enhanced functionality.
BN_T7.1_07	Dynamic and Safe RTE	*Upgradability:* Systems built according to EMC^2 principles are cheaper and more reliable to maintain than their non-EMC^2 equivalents.	**Pass:** The experience of agile development of the living lab platforms has demonstrated that EMC^2 principles and separation of concerns does lead to cleaner and better architected systems which are easier to maintain and enhance.

2.2.3 Key Learnings

2.2.3.1 TomTom

The key lesson learned during the EMC^2 project is that current and next-generation consumer-grade mobile (personal navigation) devices are not reliable enough (in terms of computation and sensing capabilities) to deliver a viable Lane navigation product experience.

As a result our research focus within EMC^2 was shifted from lane navigation for consumer devices, incorporating sensor observation software to source data for map making to localization technologies for autonomous driving cars with lane navigation as a lead test application. For this development platform NVIDIA’s Tegra platform was selected because of its powerful GPU multi-core architecture optimized to run perception technology efficiently in an in-car embedded system.

The research in EMC^2 has accelerated the decision within TomTom to develop a map- and localization product roadmap within TomTom for autonomous driving and setting up a strategic partnership with NVDIA on embedded system development for the AD market.
The concept, as proposed in EMC^2, of an in-car sub-system for perception and localization with a HMI sub-system running on an infotainment (mobile device), is still viable.
Reliable lane accurate positioning and lane accurate maps are still under development. The fusion, as addressed in EMC^2, of multiple localization sources is paramount. Promising techniques are those who directly localize the vehicle in a map using real-time lane marking information or 3D point cloud data. Next generation maps will support this by having additional layers on top of the layers which are currently used for route planning. Envisioning the availability of a suitable platform and reliable lane accurate positioning and lane accurate maps, several HMI concepts are researched and presented.
Extensive effort is put in making existing and novel components available in RTMaps middleware with standard interfaces, such that they can be integrated with the components of research partners. Furthermore, a first version of the third-party SDK has been released by TomTom.

2.2.3.2 NXP

NXP-NL has participated with ITS-G5 and platforms for vision and automated driving in this workpackage. With its ITS-G5 subsystem, NXP has developed and experimented with concurrent and time-synched use of the 2 radios available in its subsystem with data-streams of various content: control-messages, audio- and video-streaming. One channel is used for the broadcasting of control messages. The same channel is used for audio-streaming. Video-streaming takes place in another channel using unicast transmission. Low-level retransmission of packets on link-layer-level in case of packet-loss enable an excellent video experience. For the total system, the resulting delay is just msecs and the robustness of link has been proven to be very high. This ITS-G5 subsystem is part of an ADAS system which integrates the information into a control-loop for automated driving in truck-platooning. Finally, Decentralized Congestion Control has been implemented to dynamically manage the load on a wireless channel (according to the prevailing standards). NXP-DE also participated with its digital radio technology which was successfully demonstrated at last years annual review.

2.2.3.3 Technolution

Technolution developed the robotic model vehicles (in collaboration with Denso & TUE) and the multi-core control platform in collaboration with TomTom within the programme, alongside the implementation of various wireless technologies (802.11p, 868MHz & UltraWideband) in support of it. The organisation has developed significant new competence during this project, representing its key learnings, a non-exhaustive list includes;

• New Robotic Competences: The robotic sensing, computation and control activities required for the ADAS application have lead directly to the development of extended robotic competences and new business opportunities including an initiative to develop a high speed control (both motor and general robot) strategic competence.

• Multi-Core Competence: Development of a new multi-core CPU competence, specifically targeting NXP CORTEX-M4/M0 CPUs and assessing their suitability for distributed control applications.

• New Radio Competence: Development of 802.11p integration alongside 868MHz for CAM/DENM style messaging in the model vehicles and UWB for high-speed ranging and V2V/V2I communication.

• Adoption of new agile investigative techniques: The development of a new ‚what happens if we try this’ mindset within the company in a manner that remains compatible with commercial constraints (i.e. with the application of techniques such as time boxing to contain risk). This has directly influenced our business and let to new opportunities outside of the core EMC^2 activity (specifically in the long range radio, additive manufacturing and robotic control sectors).

• Mixed Criticality and Hierarchical Control Competence: Layered control competence with system partitioning to minimise safety-critical work item requirements.

In addition to these items, numerous other second order learnings have also been made (the use of new tool chains, discovery of new vendors, development of new techniques and algorithms directly applicable to primary line of business etc.), and new business has been derived.

2.2.3.4 DENSO

For DENSO the key learnings were in the field of migrating single-core software to multi-core processors. First of all, it could been demonstrated that even control intensive and real-time critical software like an engine management system can be efficiently parallelized and implemented on a multi-core system. This technology offers a huge potential for more available performance for next generation applications and for significantly reduced power consumption. On the other hand, it also turned out that the parallelization has to be done very carefully and in different situations different parallelization methods need to be applied. Otherwise, the usage of multi-cores may even decrease the processing speed of an application.
The end result of this work is a system where new software (for example, GLOSA or Speed Control) can be dynamically loaded and removed from an ECU while the system remains in active operation, representing a step forward in the art.
Because of the complexity of the parallelization of real-time single-core legacy software DENSO believes that for this much better tool support is required for a wide use of this technology. DENSO will further work on this topic and will drive the development of such tools.

2.2.3.5 TNO

TNO extended the scenario classification application that enables more efficient testing from an offline algorithm to an algorithm that runs on the test vehicles while driving. This application receives information from a 360° world model constructed by advanced multisensor fusion techniques. To explore the possibilities for deploying different mixed criticality applications on a multicore platform a safety critical cyclist AEB application is added to the same hardware platform. To ensure safe operation a safety-executive software architecture is adopted: a health monitor checks whether or not all modules run as expected and switches from the nomial mode described above to a safety mode if needed. More details about the architecture, the two different modes and the different applications can be found in Appendix 1.

TNO strongly believes in the need to adopt more realistic and extensive simulations for testing increasingly complex ADAS in relevant scenarios. In order to determine which scenarios are relevant for (safety critical) ADAS and in the context of BN_T7.1_01, TNO adopted a scenario classification algorithm that extracts scenarios from real world driving data as one of the applications limits test costs while improving the representability of the tests.

Regardless of the application, the sensor layer that reads sensors has to be real-time, to guarantee that data is forwarded in time, thus to avoid variation in the processing of data (BN_T7.1_03). This is achieved by adopting the Xenomai real-time Linux framework.

2.3 T7.2 UC Highly automated driving

2.3.1 Abstract

Preventing accidents and automation of tasks in progressively more complex driving situations requires innovation in perceiving the environment, the vehicle-state, and the reasoning about them. This can be done by the use of multiple sensor technologies such as radar, computer vision, LIDAR or even ultrasound, complementing and reinforcing each other, and communicating over a high bandwidth reliable network. The optimal use of such a huge amount of heterogeneous information requires innovations in both the involved component technologies and in the system architecture concepts. It is then mandatory to propose tailored solutions that take into consideration the heavy computational load imposed by such systems, and to adaptively combine it with the existing and upcoming platforms to significantly reduce the number of ECUs that would otherwise be required.
The main objective of this Use Case was to investigate, implement and evaluate a system architecture that best exploits the potential of existing technologies around highly automated driving. To that end, the

EMC^2 architecture and tools should enable the scheduling of time-critical and less time-critical high-performance functionalities on the same system, which will be tested in simulated driving scenarios. An urban commuting scenario has been identified, where safety for driver/passengers and predictability for other road users will have to be guaranteed. Both simulation experiments will be made to show the potential of multi-core new service oriented architectures in the context of highly automated and connected driving.

From this main goal, several specific areas of interest can be highlighted:

- Integrate and validate functions to enhance both connectivity-based services and situation awareness capabilities of highly automated driving systems.
- Explore the advantage of modeling/simulation tools from different perspectives:
    - Modeling behaviors of hierarchical, compositional, and executable components within the SOA of a multi-core target.
    - Setting up a simulation framework that permits the validation of specified functionalities in the designed embedded system (under a HIL scheme) within a realistic microscopic traffic simulation environment,
- Investigate the possibilities of EMC2 architecture and tools for highly automated vehicles on heterogeneous platforms (combining multi-core CPUs with GPGPUs or FPGAs on MPSoCs)
- Properly model mixed criticalities both of the whole chain value and within individual software components
- Explore the possibilities and limitations of both existing and enhanced embedded functionalities with respect to AUTOSAR SW compliant components.
- Analyze multi-sensor perception limits (number and nature of sensors) for a specific target and real-time constraints.
- Optimize the available resources to guarantee the critical components to be safely scheduled within the runtime environment in any operation mode (including graceful degradation modes).

2.3.2 Evaluation results

The table below shows the evaluation results regarding the fulfillment of the high level requirements that have already been identified for this use case.

Table 3: Evaluation of results regarding high level requirements

Req. ID	Short description	Evaluation plans and activities towards evaluation	Assessment of requirements fulfillment.
HL-REQ-WP07-001	Modeling and simulation environment	The objective is to explore at what extent high level functionalities can be assessed within a modeling and simulation environment for ADAS. To that end: • Some relevant metrics taking into consideration performance (computational cost, required memory, WCET and deadlines fulfillment) and power will be identified • World-time synchronization (bandwidth vs latencies) between traffic and ADAS simulators will be assessed. • Component-based modelling methodology will be thoroughly assessed to quantify the gains in terms of productivity (development cycle) and flexibility (reusability of components) • The behaviour of Hardware-in-the-loop simulation will be compared to SIL in terms of portability, separation and integration.	• the interface implementation shows that ADAS and traffic simulator have negligible latencies (<5 ms) with a state vector big enough to synchronize the behavior of around 20 moving elements • The joint work between IXION and UoMAN has shown that component based modeling, provided by X-MAN, is possible while fulfilling with timing constraints, concurrency and specific HW descriptions. • The baseline components for multi-sensor perception programmed in C++ under Linux have been successfully ported to 2 MPSoCs, orchestrated using Xenomai

Req. ID	Short description	Evaluation plans and activities towards evaluation	Assessment of requirements fulfillment.
HL-REQ-WP07-002	Resource optimization	The objective is to investigate dynamic optimization mechanisms of computational resources, according to the performance requirements without putting quality/security at risk. This high level requirement will be evaluated in the light of the following uncertainty considerations • Perception uncertainty, both in terms of deadlines fulfilment (soft-real time components) and of the accuracy of the outputs and its propagation effect in subsequent components (mainly in risk assessment) • Graceful degradation of positioning in complex scenarios, and its consequences in risk estimation and decision-making components • Mixed-criticalities management when perception and localization integrity is low, and its effect on human-in-the-loop decision-making mechanisms	• The multi-sensor perception system (combining LIDAR and stereovision) has been implemented on the Tegra K1 combining multi-core CPU and GPGPU. The acceleration achieved with respect to the original design is above x20, with extremely similar accuracy • Graceful degradation mechanisms of Localization & mapping have been implemented and tested using Xenomai to consider resources re-allocation • Stereo vision occupancy grid algorithms have achieved a 98% acceleration in execution time per frame after the resource optimization step. Processing time improvement achieved:_ from PC (CPU): 63.5 s. per frame, to Jetson (CUDA->GPU): 1.1 s. per frame • LIDAR data processing (Cartesian occupancy grid + BOF (Bayesian occupation filter) + clustering) algorithms have experimented a 99% reduction in its execution time per frame after acceleration of algorithms. Processing time improvement achieved: from PC (CPU): 40.5 s. per frame, to Jetson (CUDA->GPU): <15 ms. per frame
HL-REQ-WP07-003	Parallelization and classification	The objective is to exploit the potential of multicores to enhance the performance of specific components. The evaluation of this requirement will focus on the ability to recognize and classify computing structures of application source code that may be suitable for parallel execution in • occupancy grids for stereo-vision based perception and • sequential importance sampling for risk assessment	• The probabilistic occupancy grid approach, unfeasible on standard platforms, can after a deep study, design and implementation, be run at the frequency of the slowest sensor (Lidar) • A first design and implementation considering the parallelization opportunities of particle filters has been successfully carried out on GPU for desktop.
HL-REQ-WP07-004	Effective portability	The objective is to investigate at what extent software components can be HW independent, or otherwise, how portable it is, and as a result, how flexible is the code of the developed functionalities. A special focus will be put in timing analysis and memory resources within two differentiated activities: • Analyze graceful degradation of critical components within a mixed criticality setting	• Algorithms have been ported to GPU (Jetson) for acceleration. The achievements have been measured in terms of improvements in processing time and energy consumption from algorithm’s PC version to Jetson version. • HW used: PC (x86-64, CPU i5 3330, 4 cores, 3.00 GHz, Jetson TK1 (ARM CPU 4 cores,2.32

Req. ID	Short description	Evaluation plans and activities towards evaluation	Assessment of requirements fulfillment.
		• Investigate the consistency of various sensor interfaces (in particular comparing between simulated and real sensors)	GHz, GPU 192 cuda cores, 852 MHz) • The use of X-MAN, which relies on a component-based architecture where target HW is explicitly considered, has permitted to reduce development costs associated to portability and maintainability. • An implementation of graceful degradation mechanisms has been carried out and successfully tested for Loc&Map sub-system. • Drivers for PCIe, USB and Ethernet developed and tested for HIL, consistent with the final setting on real vehicle
HL-REQ-WP07-005	Harmonization of development tools	The objective is to facilitate a seamless integration of the tools for designing, developing and validating components, This requirement will be evaluated at functional level by comparing the real experiments and the simulation HIL environment, including • Verification of the coherence between the overall Traffic/ADAS simulation environment and the trials with real cars.	Integration and evaluation of the interface between SUMO and Scaner. The selected local area properly synchronizes the evolution of the vehicles in both simulation frameworks. In a first step this area has been set static to avoid the dynamic variability of the data to be interchanged
HL-REQ-WP07-006	Integration in heterogeneous platforms	The objective is to explore methods and tools able to increase confidence (reduce risk) on problems being observed at integration time: • Investigating how heterogeneous platforms can be properly introduced in the component-based modeling platform • Assessing how the new Hybrid SoCs can be optimized from the HIL simulation platforms • Exploring the potential problems related to the introduction of multiple sensors with high bandwidth (stereo-vision, multiple-layer LIDARs)	• The combined use of X-MAN and Art2kitect, has permited to partially describe and explicitly consider heterogeneous platforms in the component-based modeling tool • The implementation using the Tegra K1 has permitted to obtain a first assessment of the relative computational cost (memory, CPU) of LIDAR/stereo and LIDAR/LIDAR fusions for occupancy grid generation.

2.4 T7.3 Design and validation of next generation hybrid powertrain / E-Drive

2.4.1 Abstract

While the main goal of providing an industrial demonstrator platform of next generation hybrid powertrains and e-Drives for the evaluation of different technical EMC² project innovations was further focused and pursued, additional emphasis on highly connected powertrains and approaches towards connected autonomous driving features were put.

To that aim, the upcoming paradigm shift for SoA approaches for highly connected cars and the future needs for dependable system (re-)configuration at runtime have also been target for further evaluation in UC 7.3.

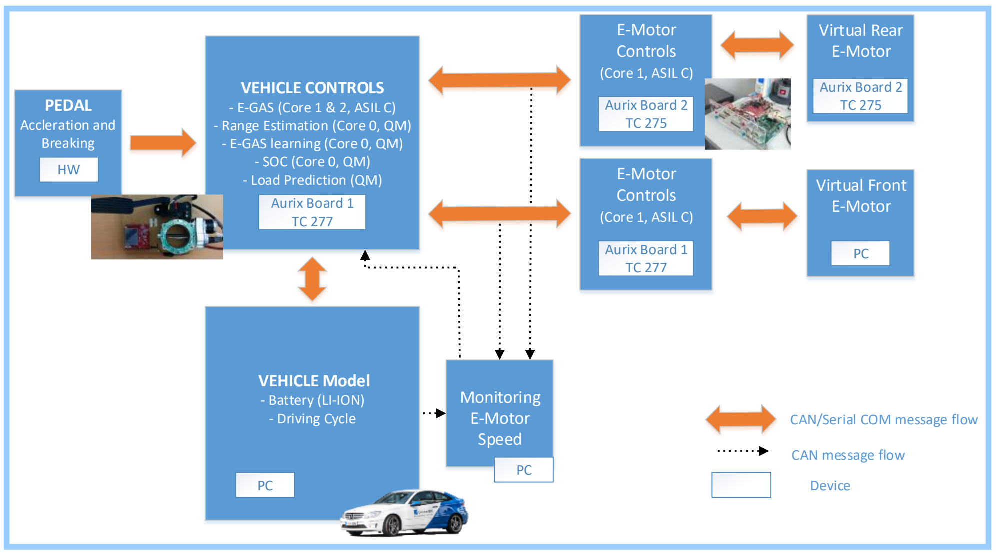


Figure 1: Demonstrator Platform of Next Generation Hybrid Powertrains and e-Drives (UC7.3)

The final project phase was related to the consolidation of solutions based on the common demonstrator platform (depicted in Figure 1).

Porting of applications to multi-core environment
To understand the paradigm shifts while porting an (existing) application from a single core microcontroller to a multi-core platform the architecture of the prototype was designed to be highly decoupled from the underlying hardware. The prototype was developed with respect to object oriented software concepts and in cooperation with the well-chosen architecture and design, the prototype is not solely dedicated to a predefined setup, but can easily expanded to a more sophisticated E/E architecture.

Multi-core tooling platform and development environment
The integration and validation of more than one software stack on a multi-core device requires new methodologies with respect to core to core communication, core synchronisation or access to shared resources and makes special demands to the toolchain in use. The development environment was implemented in a way to support distributed development environment (differned vendors using certain tool(s) form the toolchain for their development), simulated, tested and integrated on an AURIX TC277 controler.

Data exchange and communication strategies
New solutions for configuration, management, communication and verification of data exchange using automotive communication networks between multi-core controler units and development and test systems have been analyzed. The AVL in house software framework “Arte Core framework” was used and extended in order to support these required functionalities.

Safety and security co-analysis and development support
To support dependability feature development UC7.3 provides two specific use case scenarios which deal with (a) certification of functionalities via ‘Safety Contracts’ and (b) supporting of ‘Safety and Security Cases’.
The ‘safety and security case scenario’ supports proofing of safety and security during the development and deals with the question how to integrate safety and security aspects in the development process. This focuses supporting the demonstrator development on process management (WEFACT), process execution (EPF-C), and regulation and argumentation (OpenCert) base.
The ‘safety contracts’ approach allows multiple applications (developed by different manufacturers) to be certified in their final configuration (i.e. combination of applications and platform) based on a contract based runtime assurance approach, designed at development time and evaluated at runtime. The purpose tool supports safety engineer to build syntactically-correct contracts and verify their compatibility.

Ontology-based Runtime Reconfiguration
Ontology-based runtime reconfiguration (ORR) increases the availability of the system by exploiting implicit redundancy (mechanism has been developed in WP3). ORR adapts the system's structure (adding, removing or reconnecting components) to substitute failures by implicit redundancy.The approach has been successfully implemented and evaluated in the context of the UC7.3 demonstrator platform.

SoA approach
The service-oriented architecture (SoA) approach divides the system into several components, each implementing various services. Each service consumes and provides properties that are valued attributes of the system (e.g., vehicle speed). The relationships between system properties are modeled in a knowledge base (refered as ontology). Failed components can be substituted by exploiting the implicit redundany modeled in the ontology. An exemplary ontology in the context of the UC7.3 demonstrator platform has been composed and the methodology to create the domain knowledge has been demonstrated.

System (re-)configuration at runtime
A Runtime Mediator (software designed to verify the ConSerts M contracts at runtime) was developed and integrated with the UC7.3 demonstrator. This enables the possibility to check, through contracts, if the software composition can deliver its services with enough safety reliability before being deployed onto the target platform (e.g. for dependable over-the-air updating features). The development of the Runtime Mediator and the enhancement of the capabilities of the contract-language were outcomes of this project phase.

2.4.2 Evaluation results

A detailed evaluation of results can be found in Appendix 4 under 7.3 Evaluation results.

Table 4: KPI and related Achievement Measures

ID	Key performance indicator	Measurements method	Explanation on achievement
KPI29	Number of control units	Reduce the total number of ECUs in next-generation E/E architecture for commercial vehicles by 20% with the respect to the current-generation E/E architecture, assuming same level of functionalities and features realized.	UC7.3 specifics: UC7.3 provides a generic automotive E/E board net platform which enables the integration of new architecture concepts such as Service-Oriented Architecture (SOA) and provides an integration platform for mixed-critical SW stacks. UC7.3 demonstrator implements 5 mixed-critical SW stacks from 3 different suppliers on one ECU
KPI30	Degree of integration of applications of multi criticality levels	Increase integration of functionalities with different criticality levels within a single ECU by 20% with respect to the current level of functionality integration by exploiting the potential of multicores.	UC7.3 demonstrator implements 5 mixed-critical SW stacks from 3 different suppliers on one ECU
KPI31	Number of dissemi-nation occasions	Representatives from WP7 present their results in at least 3 major European conferences and/or exhibitions.	UC7.3 related publications at: • INDIN 2015 • ERTS² 2016 • EuroSPI 16

KPI32	Efficiency of control algorithms	Increase efficiency by in average of 20% of existing automotive mechatronic elements by porting and integration of more complex (resource and time consuming) algorithms on multicore technology. Efficiency in the context of WP7, must be refined according to the use case considered. Efficiency can be defined e.g., as vehicle range (efficiency of battery module) for electric vehicle, or as confidence in the system for highly automated driving (efficiency of the algorithms to take the good decision).	The multi-core platform provides more computing resources for integration of new functionalities and / or more complex algorithms. Thus, e.g. by integrating advanced EIS analysis of individual battery cells, the vehicle range can be extended due to reduction of necessary safety margin (regarding SoC). Further a decrease of energy consumption due to advanced load prediction model and learning function can be achieved.
KPI33	Trustworthiness of ACC system architecture	Make effective the trustworthiness and, finally, the safeness specifications at concept level of ACC system architecture (starting from the state of the art prototype, the 100% of possible architecture malfunctions and related hazards has to be analyzed in the corresponding operational situations and at least more than 75% of the necessary safety requirements at concept level have to be defined).	Current safety approaches assume, that a system as well as its usage context is completely known and can thus be analyzed thoroughly at development time, which does no longer hold true for SoA approaches. As a result, a shift of parts of the safety certification activities to runtime, where the complete information can be obtained and uncertainty can be truly resolved, is focused. The integrated dependability framework invests ensure to further coalesce SW and safety engineering and strengthen the integration of these engineering domains.
KPI34	Degree of contribution to the development of sustainable mobility and transportation	Contribution to the development of a technology for sustainable mobility and transportation. This contribution will not be quantifiable in the timeframe of the project, but it will be supported by extrapolation of trends, e.g. support an initial 10% extension of the electric traction applications with respect to the actual technology based on conductive recharging, 10% CO2 reduction due to more efficient vehicle control (less conflicting actions by ADAS systems) and information exchange via C2X systems.	To achieve further benefits a further enhancement of the control functions and re-design of the current E/E architectures will soon be required. Consequently, UC7.3 focuses on enhancement and adaptation of vehicle electric / electronic architecture to better suit (a) the requirements coming from new applications fields such as higher degree of electrification and hybridization driving, and (b) opportunities coming from SOA approaches.

2.5 T7.4 Modelling and functional safety analysis of an architecture for an ACC system

2.5.1 Abstract

The primary objective of the T7.4 was to identify, implement and evaluate a methodology that would have a significant impact in reducing time to market and development cost of safety mixed criticality systems, supporting the application of the ISO 26262 [2] standard process, with dedicated procedures for guidance/control/verification of safety requirements design, having as implementation example an ACC (Adaptive Cruise Control) system.

The methodological approach started from the modelling of an ACC (Adaptive Cruise Control) system application example, covering the main steps of ISO 26262 safety cycle at "concept level". [Addressing the "concept level" is intending that the current tool chain objective does not encompass the entire ISO 26262 standard yet, but it is related to the deployment of the functional and the technical safety concepts and their verification in terms of the corresponding safety requirements (functional and technical safety requirements)].
The deliverable "D7.10 – Detailed UC7.4 Design" presented and summarized the activities related to the completion of the core methodology, of which a preliminary version was outlined in the previous "D7.09 – Preliminary UC7.4 Design".

The activities related to the use case detailed deployment have been executed in cooperation by CRF and DITEN (University of Genova), where CRF has defined and deployed the general outline of the methodology concept and the basis of the meta-model description, with respect to the ISO 26262 standard process, and DITEN has contributed to the modeling design and implementation.

The developed methodology covers the steps for the description and management of the Safety Requirements chain through the development of custom profiles [Model Driven Generation (MDG) Technologies], internal code scripting, external code libraries (Add-Ins), as depicted in Figure 2, where the numbering of the steps corresponds to the following list of actions:

Item Definition:

0) Functional requirements definition

1) System Architecture(s) outlines creation for Requirements allocation

Hazard Analysis and Risk Assessment (HARA)

2) HARA results

Deployment of the Safety Requirements (SRs) chain outline

3) Derivation of Functional Safety Requirements (FSRs)

4) Allocation of FSRs

5) FSRs ASIL Decomposition

5.1) Reporting of Functional Safety Concept (FSC)

6) Derivation of Technical Safety Requirements (TSRs)

6.1) Reporting of Technical Safety Concept (TSC)

7) Verification stage

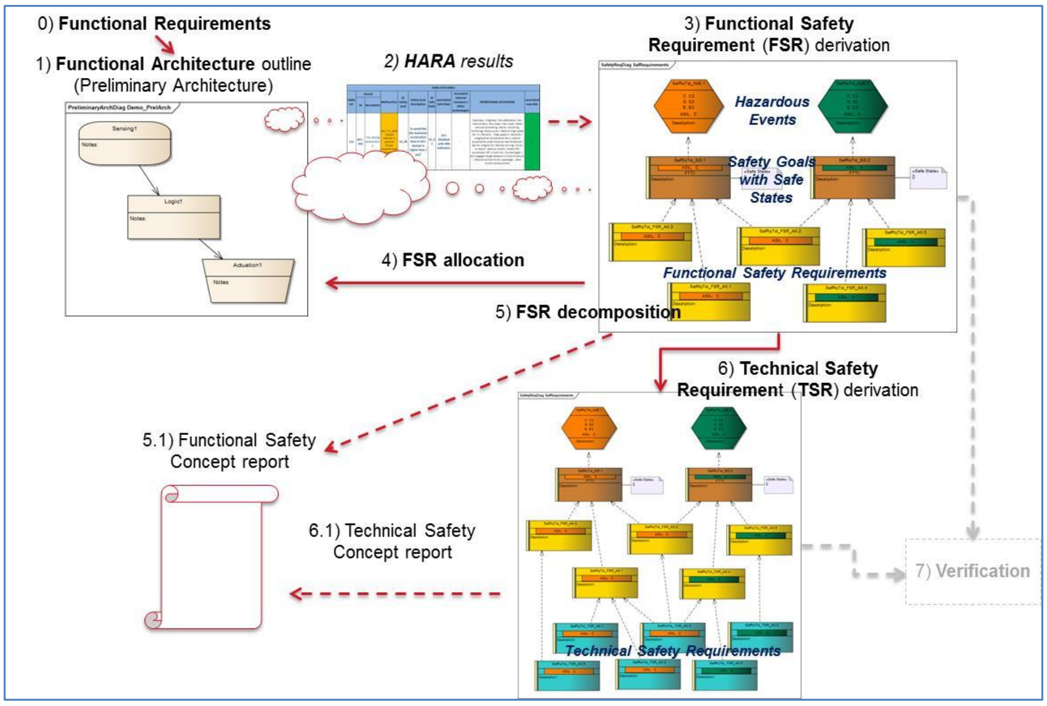

0) Functional Requirements
1) Functional Architecture outline (Preliminary Architecture)
2) HARA results
3) Functional Safety Requirement (FSR) derivation
Hazardous Events
Safety Goals with Safe States
Functional Safety Requirements
4) FSR allocation
5) FSR decomposition
6) Technical Safety Requirement (TSR) derivation
5.1) Functional Safety Concept report
6.1) Technical Safety Concept report
Technical Safety Requirements
7) Verification

Figure 2: Process implemented by the methodology (UC7.4)

The intended methodology has developed a graphical meta-language/meta-model, on the basis of a SysML approach (semi-formal language approach), focused on the ISO 26262 standard "V" cycle as represented by Figure 3.

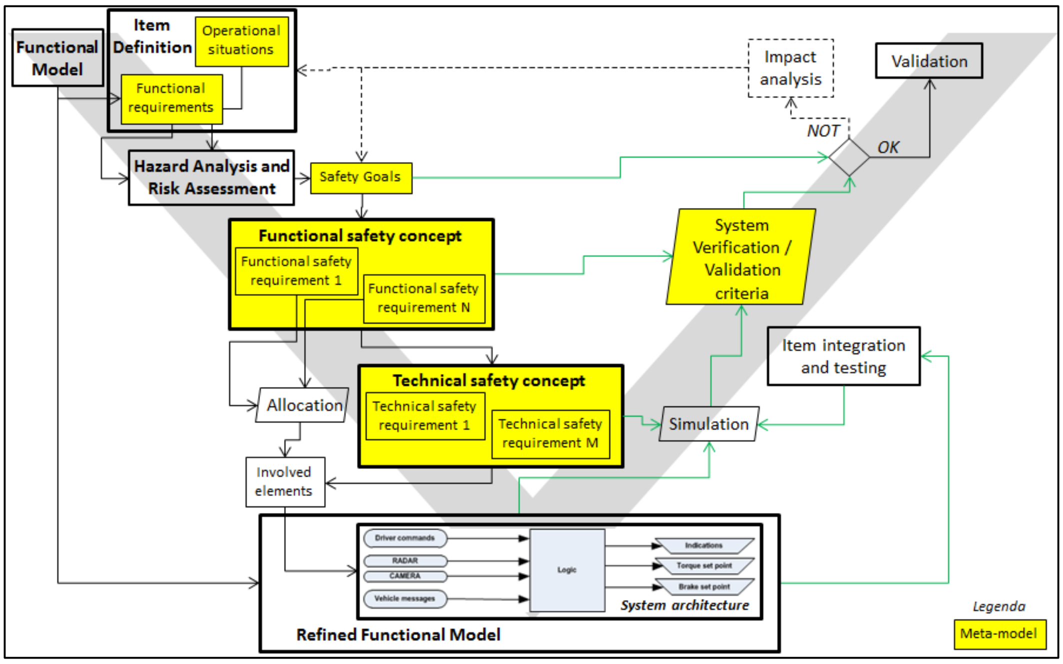


Figure 3: ISO 26262 standard "V" cycle covered by the methodology (UC7.4)

The developed methodology is integrated with an internal best practice, starting from the Office (Word/Excel) environment with dedicated templates, from which the starting points of the system development process are established as preliminary architecture and functional requirements.

2.5.2 Evaluation results

UC7.4 proposes a new approach to modelling safety mixed criticality systems and related safety requirements, for supporting the development of projects compliant to ISO 26262 standard with the use of a semi-formal language in a structured environment: custom SysML in Enterprise Architect framework (SysML/EA) has been exploited for this goal.

The modelling example of an Adaptive Cruise Control (ACC) system has been used as the basis of the evaluation, with the development of the safety requirements up to the technical safety level. SysML and EA have been used as the foundation for developing appropriate MDG technologies, with artefact profiles tailored to the aims and scope of the application. Custom toolbars, toolboxes and procedures (script and Add-Ins) have been developed and integrated into the Enterprise Architect framework for fulfilling the objectives related to the high-level requirements of the task.

The methodological/technological approach satisfies the needs of creating automatic rules to support the prescriptions from ISO 26262 standard as for the inheritance and traceability of safety requirements attributes.
The phase of safety requirements verification/testing through simulation has been explored as well, even if the methodology cannot exploit automation for this last step.

The envisaged methodology anyway, could have a significant impact in reducing time to market and development cost of safety mixed criticality systems, supporting the application of the ISO 26262 standard process, with dedicated procedures for safety requirements design guidance and control.
In these cases traceability and confidence on the managed information are crucial for reducing errors and improving the impact analysis, especially in case of eventual modifications occurred during or after the work in progress.
For the future, specific work should be dedicated to improve and automatize the links between the ISO 26262 compliant safety requirements system design and the more specific tools for testing.

2.6 T7.5 Use case Infotainment and eCall Multi-Critical Application

2.6.1 Abstract

The goal of this use case is the design and integration of a small, multi-core, mixed-criticality software platform managed by a realtime operating system (RTOS) developed in various WP3 tasks. The RTOS provides the deployment and scheduling of tasks with distinct safety-critical requirements on the multi-core hardware. This platform also provides a hypervisor that allows the RTOS to run concurrently with a paravirtualized commercial off-the-shelf (COTS) general purpose operating system with infotainment capabilities that shares resources safely (i.e., without interfering) with the safety-critical runtime environment (RTE).

The implementation of mixed-criticality applications requires strong partitioning of applications and RTE, while still maintaining good performance and compliance with the automotive standard ISO26262. To fulfil these requirements, the Freescale SABRE i.MX6 Quad System on Chip (SOC) was selected as the hardware platform, which provides a hardware enforced software containment solution already certified for automotive usage. This enabled a small virtualization software footprint by the usage of TrustZone®, a hardware enforced container by ARM® available on i.MX6 SOC.

In order to develop the multi-core multi-criticality demonstrator, we've split into sub-tasks the various steps required to design, develop and integrate the technology used in the use case's demonstrator. The following activities were performed:

- Requirements and specification: specification of business needs, followed by the high level and technology requirements;
- Safety Aspects of Multi-criticality Applications: The architectural design definition has been supported by two key safety-related activities that are required by the ISO26262 standard. Firstly, a hazard analysis and risk assessment for the identification and categorization of the hazards that can be potentially triggered by malfunctions in the demonstrator. And secondly, a safety analysis, to examine the consequences of potential faults and failures of the demonstrator functions, behaviour and design;
- Integration of HW and SW Platform to support Multi-Criticality: integration of the technology developed in WP3 and with the various components designed and specified.
- Integration between Infotainment and eCall Applications: This sub-task was originally dedicated to the integration of an "Advanced eCall" application into the demonstrator. Since the integration was not possible due to changes in the consortium composition, an alternative solution was adopted. Running on the RTOS userspace, a crash detection and notification task provides one of the use case's goals of sending and emergency notification in the event of the vehicle crash.
- Use Case Demonstrator: preparation of the demonstrator, developed in C language, with specific and optimized sections developed for use with an ARMv7a assembly.
- Evaluation: evaluation of the effectiveness of the demonstrator in delivering a multi-core mixed-criticality system, composed of an infotainment environment and a critical automotive application.

2.6.2 Evaluation results

In the table below the assessment of the KPIs for this use case is shown.

Table 5: KPIs for use case "Infotainment and eCall Multi-Critical Application"

No	Title	Description	Assessment of KPI fulfillment
KPI_T7.5_01	Mixed criticality tasks	The execution environment must support the deployment of tasks with at least two distinct levels of criticality	The use case demonstrator showcases two types of mixed-criticality. One on the RTOS where tasks can execute with different levels of safety-critical requirements and the other on a concurrent RTE running a COTS general porpuse OS. On the safety-critical RTE the space

			isolation of the non-critical and safety-critical tasks is ensured by enforcing virtual memory addressing (VMA) for non-critical tasks using the architecture supplied Memory Management Unit, the CPU's unpriviledged mode of execution and RTOS controlled page tables. Time isolation is provided by the RTOS hierarchical scheduling and secure interrupts. Communication between user and kernel is achieved through CPU exceptions for kernel system calls that handle user requests. On the non-critical RTE the same space isolation is applied although on an isolated memory space defined at boot time by the safety-critical RTOS and enforced by the Trustzone Address Space Controller (TZASC). For time allocation, the non-critical RTE is free to use the assigned cores unless the safety-critical RTE requires otherwise. In such case the safety-critical RTE signals the non-critical assigned CPU cores with a secure interrupt that ensures they return to safety-critical RTE control. This transition is ensured by the safety-critical RTOS leaving the non-critical RTE or non-critical realtime task transaction in an unknown state but retaining the safety-critical access.
KPI_ T7.5 _02	Multicore tasks	The execution environment must support the deployment of concurrent multi-core applications.	In order to be able to concurrently run applications of multiple criticality levels in a multicore platform, a server-based (or hierarchical) scheduling approach was implemented. Server-based techniques are an effective solution to ensure the temporal isolation and protection of the applications, while respecting all their realtime constraints. Servers allow for the reservation of a portion of the embedded system capacity for a specific application. Therefore it allows applications to run independently through time partitioning. The servers are allocated to the respective core. The servers have a fixed-priority can be preempted. Tasks of distinct levels of criticality can then be assigned to their respective server (time partition) on their defined CPU core, without interfering with each other's response time. Note also that the separation between the non-critical RTE and of the secure RTE is flexible in relation to the assignment of the cores for each environment. For the demonstrator, a quad-core hardware platform was selected, where two cores were assigned for each environment. The secondary core is optional, and can be activated depending on the initial configuration.
KPI_ T7.5	Secure communicatio	A communication mechanism must be available for secure	The secure inter-core communication mechanism (ICC) was developed to act as a

_03	n	data transfer between two environments of distinct criticalities	lower level communication layer that enables endpoints of distinct criticality levels to communicate with each other taking into account the system configuration, target permissions and availability. The ICC uses the hardware security and hypervisor extensions to ensure correct time and space partitioning and prevent error propagation from both safety-critical tasks as well from the non-critical RTE. The architecture, consists of a series of virtual channels where messages are delivered to their intended targets taking into account both the requesting and target's safety requirements and constraints. These targets can be platform components such the Hypervisor or the RTOS scheduler where they register handlers for message/command events, or they can be tasks that must pool for new messages to handle. The channels define the target core(s) to where the request shall be added into each priority run queue allowing messages to be delivered to multiple targets (e.g., signal the hypervisor on the non-secure assigned CPU cores to take control back from the non-critical RTE).
KPI_T7.5_04	Android	The Android Operating System shall be the runtime environment used as the Infotainment system	In order to enforce the time and space isolation between the critical and non-critical RTE, the Android OS runs within a hardware enforced software container in an unprivileged mode of execution, which is implemented by a hypervisor developed in WP3. In order to support correct execution under the mentioned technologies, the Linux kernel is modified to accommodate the necessary virtualization requirements, such as virtual device drivers as well the secure communication mechanism using the Hypervisor API. The non-critical RTE still has direct access to the hardware assigned to it, as well to CPU exception handling, meaning interrupts or exceptions from the non-critical side are handled by the non-critical RTE.
KPI_T7.5_05	FreeRTOS	The FreeRTOS shall be used as the runtime environment for the RTOS	The initial concept envisaged FreeRTOS as the preferred RTOS. However, FreeRTOS presented several limitations, which would not allow to fully fullfil the project's requirements. Some of these limitations include: lack of multicore and user task support, no device management support, no definition of groups and user permissions. Therefore, a completely new solution was implemented from scratch. And although FreeRTOS was not adopted, all the key required features for an RTOS have been achieved, and even exceeded in several aspects. For example, the implementation of state of the art solutions, both in terms of

			space isolation through the use of ARM Trustzone technology, and in terms of time isolation, through the development of an adaptation of the MC-IPC scheduling and resource sharing protocol.
KPI_ T7.5 _06	Infotainment	The infotainment system shall use the Android applications for its multimedia purposes	As mentioned in KPI 4, the COTS Android execution environment is fully supported. The ANDROID OS is a feature rich environment with great user interaction is the base platform for the infotainment system. It's a known OS with a vast amount of downloadable user applications while allowing the system designer to easily tailor it for automotive usage and safety limitations. The adaptation made to its source was limited to removing boot aspects that were already handled by the safety-critical RTOS.
KPI_ T7.5 _07	System devices	HW resources shall be managed by their assigned OS	Taking into consideration the capability of direct hardware access from the non-critical RTE provided by the hardware security extensions, the system designer is capable of defining at design time the safety-critical level and share type of each hardware device leaving to the RTOS device manager the responsibility to ensure at all times that there is no mismatch between accessing RTE and target device safety-critical level occurs. However, hardware device management in mixed-criticality systems must ensure correct time and space partitioning. The timing constraints that come from the requirements of safety-critical tasks are able to lock resources within their execution time leaving the remaining time slots (if any), available to other tasks/groups. To fullfil this requirement, the RTOS kernel provides the necessary interfaces for configuring, obtaining and releasing all managed devices. The management of the devices is achieved through the use of descriptors and access control, through the device drivers and device bus management APIs.
KPI_ T7.5 _08	eCall system	An eCall system shall be developed as a critical real time task on the RTOS runtime environment	In the original concept, the objective was to fully integrate eCall applications from partner HIB. Since the integration was not possible due to changes in the consortium composition, an alternative solution was adopted. Running on the RTOS on top of the userspace, a crash detection and notification task provides one of the use case's goals of sending and emergency notification in the event of the vehicle crash. The integration of resources to functionalities enabled the use case demonstrator to showcase resource and information sharing between two applications of distinct levels of criticality.
KPI_ T7.5	Software Containment	The distinct levels of criticality shall be addressed	One of the major aspects that has been achieved by the demonstrator is to show case

_09		using hardware assisted virtualization.	two major functionalities with distinct levels of criticality, running on the same multi-core hardware platform. The method selected to achieve software containment is the use of a hypervisor together with hardware supporting features. The hypervisor configures the hardware that in turn enforces the space and time partitioning, so that failures in the non-critical RTE cannot propagate and affect safety-critical applications, while still allowing communication between them. This is also shown in the mixed-criticality aspect of user and kernel realtime tasks that the safety-critical RTE provides. This is controlled by the RTOS kernel using the hardware's execution protection to provide separation between user and kernel space through the use of MMU and CPU modes of execution.
KPI_ T7.5 _10	Resource partitioning	The execution environment shall allow for space and time partitioning.	In order to correctly define and manage access to the system resources, each resource needs to be attached to a permission protocol and subsequent management and access API. Sharing of such resources demands a sharing protocol to be defined, one which guarantees predictable worst-case access in mixed-criticality systems. An adapted MC-IPC protocol provides the required time partitioning between resource accesses by mixed-criticality tasks in a multi-core platform with a simple data structure.
KPI_ T7.5 _11	Resource takeover	The execution environment shall support non-critical resource takeover by critical component initially determined as non-critical.	The RTOS provides a permission protocol similar to the one used by *nix systems where flags define the type of access the user, group or others can have that other components can use to enforce access policies. The main difference comes from the concept of user and group being applied instead to the task ID and safety-critical ID. This means that each task has an ID and safety-critical level attributed at design time that identifies itself as a "user" and the safety group it belongs. The remaining access attribute flags (read, write and execute) remain the same. Resource access permissions may be subject to changes at runtime in order to accommodate requests from both safety-critical RTE and non-critical RTE. This is shown in the demonstrator by revoking access to the non-critical RTE to the CPU cores assigned. The tracking of resource allocation is managed by the RTOS and for example in hardware devices it's enforced by either configuring the hardware to accept only either safety-critical or non-critical RTE requests or by accepting system calls requests from tasks with the required access permissions. Another example is system call

			access permissions where the access is verified upon each request against that call access policy.
KPI_ T7.5 _12	Inter-core communicatio n	The execution environment shall support an inter-core communication API and be able to send a signal between cores.	As already described in KPI_T7.5_03, the ICC mechanism is used to send messages and commands to targets assigned or running in any of the existing CPU cores.
KPI_ T7.5 _13	Execution monitoring	The execution environment should at least support the monitoring of CPU utilization and idle time.	Major components in a mixed-criticality platform (e.g. scheduler, hypervisor, inter-core communication and memory management) require extensive certification and must be fully tested to ensure total coverage. However, each the code implementation of each component has instructions to deal with error cases that are, in many cases, easily reachable without hardware errors. To reach those sections the components must register fault/monitoring points to specific variables to being monitored and altered in runtime in order to trigger such error conditions. These variables can range from the scheduler's jiffies to the memory manager stack corruption detection. Moreover, the following statistical data is available per component: • CPU usage (total and per-core) (Scheduler) • Task deadline failures (Scheduler) • Memory page allocation statistics (Memory Manager) • Number of memory access exceptions (Memory Manager) • Interrupts received between a specific time delta (Interrupt Manager) • Interrupts target (Interrupt Manager) • Number of messages in each core queue (Inter-Core Communication) • Online status (Infotainment System).
KPI_ T7.5 _14	Core affinity	The execution environment should allow binding a function to a specific core.	Core affinity is extensively supported. The affinity of all the environment executable entities and events are bound to a core (or group of cores) by default, such as: • Tasks are assigned to a specific core; • The safety-critical RTE may be assigned to a determined number of available cores; • Likewise, the non-critical RTE may be assigned to any of the available cores. • Interrupt handling is CPU core bound but reassignable at runtime.

More details about the use case are available in Appendix 5: T7.5 UC Infotainment and eCall Application Multi-Critical Application.

The EMC^2 deliverable **D7.12 – Detailed UC T7.5 Design, first prototype and preliminary evaluation result** presents more detailed information about the use case design of the T7.5.

2.7 T7.6 Next Generation Electronic Architecture for Commercial Vehicles

2.7.1 Abstract

The goal of this use case is to identify, apply and evaluate technologies, methods and tools to harvest the potential of multicores for mixed-critical applications in the automotive industry. Main topics of interests are to explore the full potential of multicores based on the needs and requirements of the next generation Electrical and Electronic (E/E) architecture for trucks. In particular, this use case has focussed on:

- Efficient E/E architecture, including communication aspects (logical and physical architectures, mapping between logical and physical architectures, qualitative and quantitative evaluation of alternative architectures) based on multicores.
- Efficient partitioning and allocation of functionalities as well as software on multicores, including separation among elements of different criticality levels and management of shared resources.
- Applicability of Service oriented Architecture (SoA) and run-time optimizations for real-time safety-related systems based on AUTOSAR [1].
- Improvement of dependability (functional safety, reliability, fault tolerance, robustness, availability, security) by using multicores along with support for scalability, flexibility and adaptability.

Requirements consists of both functional and non-functional requirements whereas constraints may encompass cost, legacy, technological trends, available hardware resources (e.g., computation power, memory, communication bandwidth) and market trends (e.g., unanticipated new features). The objective is then to identify an E/E architecture that (a) fulfills the functional and non-functional requirements, (b) supports scalability, flexibility, adaptability and reusability, and (c) finds the best possible alternative by applying multi-objective optimization, considering the constraints. Note that this use case consists of several main topics of interests towards the common goal of exploiting the potential of multicores. To achieve the objective and make progress beyond state-of-the-art, the following innovation activities have been performed within the T7.6:

- Metrics for systematic evaluation of electronic architecture alternatives. Embedded vehicle architecture concepts based on computation nodes and generic I/O nodes
- Efficient methods and tool support for allocation and scheduling of software on multicore ECUs.
- Develop tools and techniques to exploit multicore systems properly taking into consideration resource sharing issues among cores. Methodologies for enforcing strict isolation in a multi-core system.
- Service oriented Architecture (SoA), dynamic configuration of services and their optimization during runtime, considering real-time requirements.
- Methodologies for analyzing multi-core ECU configuration with respect to compliance with safety requirements. New algorithms for efficient detection of faults on multi-core platform and new methodology of fail-safe or fail-operational system development.

Relevant standards and specifications such as ISO 26262 and AUTOSAR are central to the use case with respect to not only considering present requirements on the E/E architecture and multicores but also adapting these standards to state-of-the-art in multicore systems with mixed-criticalities.
In appendix 6 more information about the use case is shown.

The EMC^2 deliverable **D7.14 – Detailed UC T7.6 design, first prototype and preliminary evaluation result** presents more detailed information about the use case design of the T7.6.

2.7.2 Evaluation results

In the table below the assessment of the KPIs for this use case is shown.

Table 6: KPIs for use case "Next generation electronic architecture for commercial vehicles"

No	Title	Description	Assessment of KPI fulfillment
KPI_T7.6_01	Reduction of number of control units	Reduced total number of ECUs in next generation E/E architecture for commercial vehicles by 20% with the respect to the current generation E/E architecture, assuming same level of functionalities and features realized.	We have in ST7.6.2 of the use case investigated how to reduce the number of control units by merging single core software into multicore control units. This KPI is met.
KPI_T7.6_02	Definition of architecture evaluation methods	Defined systematic evaluation methods to compare alternative E/E architectures to identify the most suitable one that meets a given set of requirements as well as supports scalability, flexibility, reusability and adaptability. Such evaluation methods should include embedded network communications.	A set of evaluation criteria have been defined. A reference evaluation of existing architecture has been made. However, it was concluded that we could not produce all data needed, so it is partly evaluated based on estimates and subjective judgements. The approach to go further is to use the criteria as driving criteria for conceptualisation of architecture, rather than actually evaluating new architecture concepts. KPI is partly fulfilled.
KPI_T7.6_03	Definition of concepts, methods and tools for partitioning and allocation of software	Defined concepts, methods and tools for efficient partitioning and allocation of functionalities as well as software across the multicore ECUs at the E/E architecture level and across the processor cores within a particular ECU to maximize performance and resource utilization.	A tool for allocation and scheduling of single core software om multcore control units has been developed together with EMC^2 partner DTU. The tool considers a set of criterias such as communication between cores, load balancing, criticality level etc. In parallell a visualization tool has been developed in order to show different software dependencies such as signal paths. The tools have been applied for an internal real case and are proven as feasible and are fulfilling this KPI. A concept to provide an ISO26262 compliant BSW has been developed together with a tool to configure and generate Autosar communication between cores. This has also been developed within the scope of ISO26262 to be able to support mixed criticality between cores.
KPI_T7.6_04	Definition of methodologies for dependability enhancement	Defined methodologies and guidelines for utilization of available resources of multi-cores to improve product quality and safety by enhancing dependability (functional safety, reliability, fault tolerance, security, and availability) both at the E/E architecture level and at the	The scope of ST7.6.4 has been decreased due to resource issues. However, a guideline for robustness evaluation has been developed. The KPI is just partly fufilled.

		ECU level.	
KPI_ T7.6 _05	Increase of the degree of integration of applications of mixed criticality levels	Increase integration of functionalities with different criticality levels within a single ECU by 20% with respect to the current level of functionality integration by exploiting the potential of multicores.	The mapping tool from DTU assumes an AUTOSAR platform and relies on the AUTOSAR supported mechanisms for ensuring freedom from interference. In case a non-AUTOSAR platform is used, the same kind of support is provided by an alternative solution. This includes E2E protection, memory protection and execution time monitoring. KPI is fulfilled. Arccore have provided a concept on how to build a mixed criticality AUTOSAR platform VIF has developed new schedulability-analysis methods for AUTOSAR-systems with timing-protection capabilities. These allow evaluating mixed-critical systems with several timing-uncertainties, and thus enabling to increase the degree-of-integration.
KPI_ T7.6 _06	Definition of communications strategies	Defined strategies for Inter-ECU (at the E/E architecture level) and intra-ECU communication (network topology, protocol, bandwidth, delay, technology).	In the architecture alternatives the aim is to use as much general purpose communication standards as possible, e.g. standard Ethernet. A specialization is the Automotive Ethernet which is an alternative. In terms of topology a switch as the central unit with point to point connections to attached control units is one possibility that we have been using further. In the SoA demonstrator developed in this use case we are evaluating the CoAP, Constrained Application Protocol, which is used as application protocol on top of UDP/IP/Ethernet. KPI is partly fulfilled. Arccore have demonstrated a means of inter-core communication that supports mixed criticality within AUTOSAR
KPI_ T7.6 _07	Definition of Service-oriented architecture concepts	Defined concepts including viability and applicability of Service-oriented Architecture (SoA) for real-time functional safety-related automotive systems.	A study of the impact of introduction of SoA in a truck architecture has been done and corresponding pros and cons has been derived. Arrowhead SoA framework is implemented using SoaML modelling standard. The software structure used at Volvo is followed. KPI is partly fulfilled since real time and functional safety behaviours are not fully assessed.
KPI_ T7.6 _08	Definition of runtime optimization concepts	Defined concepts including viability and applicability of run-time optimizations for real-time safety-related automotive systems.	This KPI has not been evaluated.
KPI_ T7.6	Definition of concepts and	Defined concepts and recommendations to (a)	Arccore has developed a Complex Device Driver to support inter-core

_09	recommendations for AUTOSAR support for multicore, SoA and runtime optimization.	improve multicore support for mixed-critical systems in future releases of AUTOSAR as well as other relevant standards and specifications, and (b) suggest required changes in AUTOSAR to support SoA and run-time optimizations.	communication and an additional tool to split the RTE communication across cores. This can be done on top of the existing Autosar standard without the means of changing the standard.
KPI_ T7.6 _10	Definition of concepts for mixed OS support	Defined concepts for mixed OS support in one and the same ECU.	A study was made using ArcCore AUTOSAR OS running alongside Linux using the Xen open source hypervisor on Dual Core ARM A7 hardware. The concept of using hypervisor to support multiple operating systems in different use cases has been studied, with the primary focus on reducing boot time when both AUTOSAR OS and Linux are running in the same CPU. The KPI is assessed to be fulfilled.

3. References

[1] AUTOSAR www.autosar.org

[2] ISO26262 Functional Safety for road vehicles

[3] SAE publication http://papers.sae.org/971790/

[4] Panic, Milos; Kehr, Sebastian; Quiñones, Eduardo; Böddeker, Bert; Abella, Jaume and Cazorla, Francisco: RunPar: An Allocation Algorithm for Automotive Applications Exploiting Runnable Parallelism in Multicores. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS), New Delhi, India, October 2014

[5] Kehr, Sebastian; Quiñones, Eduardo; Langen, Dominik; Böddeker, Bert; Schäfer, Günter : Parcus: Energy-aware and Robust Parallelization of AUTOSAR Legacy Applications. 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). IEEE. 2017-04.

4. Abbreviations

Table 7: Abbreviations**Abbreviation**	**Meaning**
ACC	Adaptive Cruise Control
ASIL	Automotive Safety Integrity Level
AUTOSAR	Automotive Software Architecture
BN	Business Need
BCM	Body Control Module
BSM	Braking System Module
BSW	Basic Software
EAP	Enterprise Architect Project
ECM	Engine Control Module
ECU	Electronic Control Unit
FSC	Functional Safety Concept
FSR	Functional Safety Requirement
FTTI	Fault Tolerant Time Interval
GLOSA	Green Light Optimized Speed Advisory
HARA	Hazard Analysis and Risk Assessment
HWSR	Hardware Safety Requirement
ISO	International Standardization Organization
KPI	Key Performance Indicator
MDG	Model Driven Generation
QM	Quality Management
RTE	Run Time Environment
SR	Safety Requirement
SWSR	Software Safety Requirement
SysML	Systems Modelling Language
TSC	Technical Safety Concept
TSR	Technical Safety Requirement
UC	Use Case
WP	Work Package
µC	Micro-Controller

5. Appendix 1: T7.1 UC ADAS and C2X

Within this Appendix slightly more information is presented about the activities of each of the partners within the programme. The interested reader is directed both to D7.4 and the exploitation report and the final review demonstrations for further information about the material presented in this appendix.

5.1 Partner Activities

5.1.1 NXP

NXP has developed a robust communication system for Platooning communication. The design provides platooning messaging, audio and video, diagnostics and a check on the authenticity and data-integrity of the platooning messages.

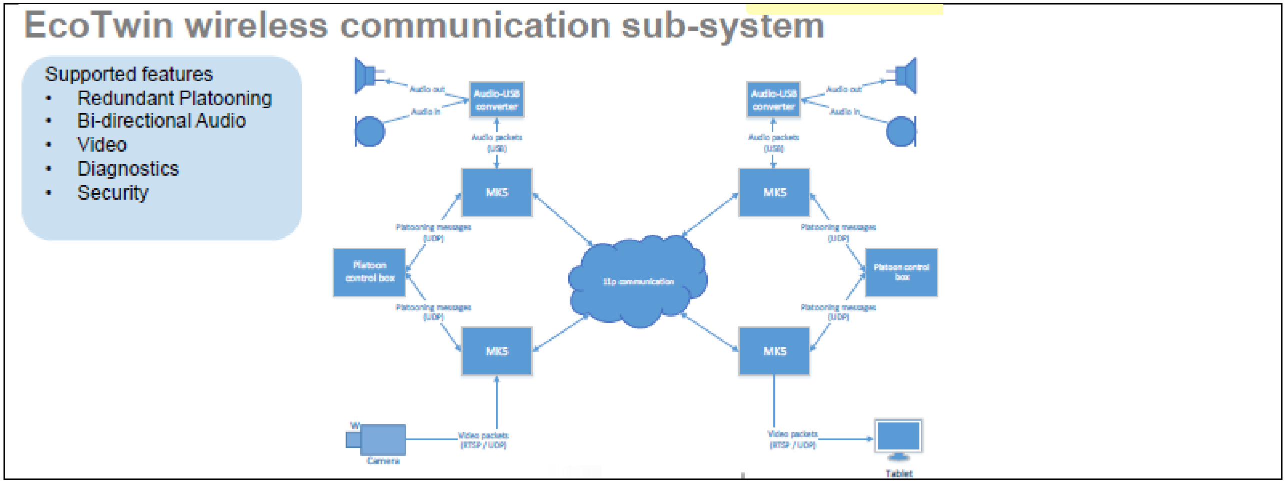


Figure 4: High level Design of the wireless communication subsystem based on NXP ITS-G5 communication solution

Through a redundant design for the platooning messages a very high reliability is achieved, as its first test on the open road has shown in the following figure;

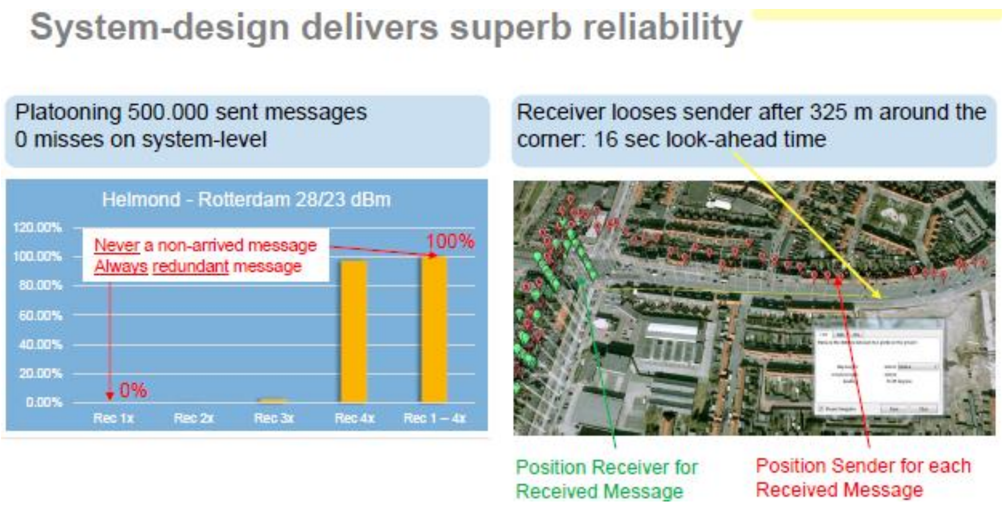
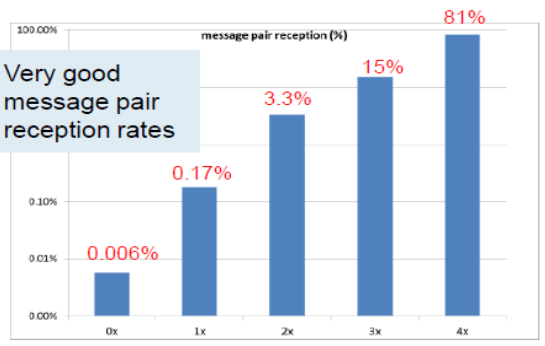


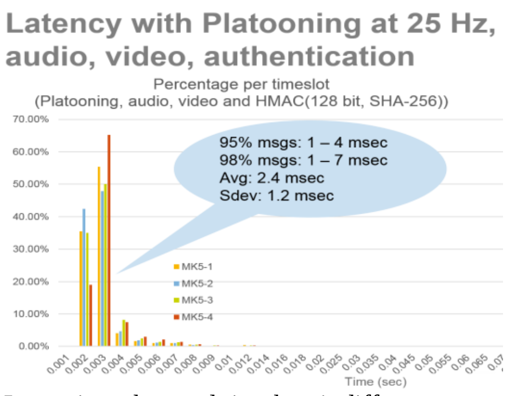
Figure 5: Reliability of wireless communication subsystem in first on the road tests

Reliability and sensitivity of the platooning system in its first prototype set-up on the road.

After finalizing the system development, the system was deployed in platooning trucks as part of the EU Truck Platooning Challenge 2016, in a consortium consisting of DAF, TNO, Ricardo and NXP. Actual results show a high reliability as well as a sufficiently low latency, allowing platooning at a short distance.



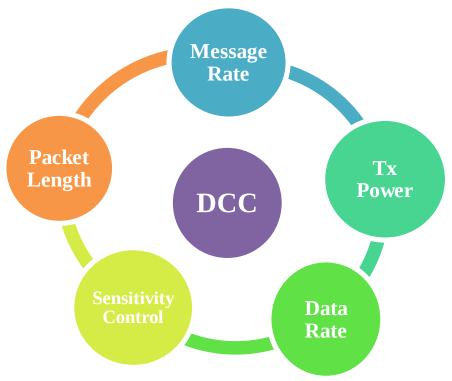
Message reliability in platooning trucks (data courtesy TNO)



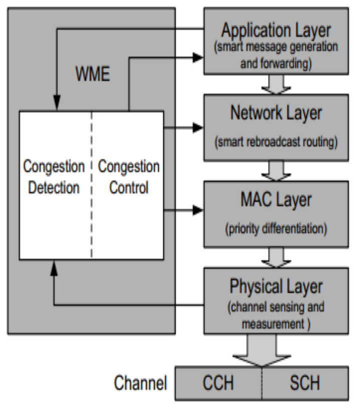
Latencies observed in the 4 different communication paths for platooning.

Figure 6: Quality of Service of wireless communication subsystem carrying audio, video and platooning control messages during platooning.

Finally, in anticipation of congestion in a wireless channel, decentralized congestion control has been implemented in the principal design. Through various means the load in a channel can be dynamically adjusted, measured through a channel busy ratio. The technology and methods for this were developed in WP3, and applied in the wireless design for platooning.



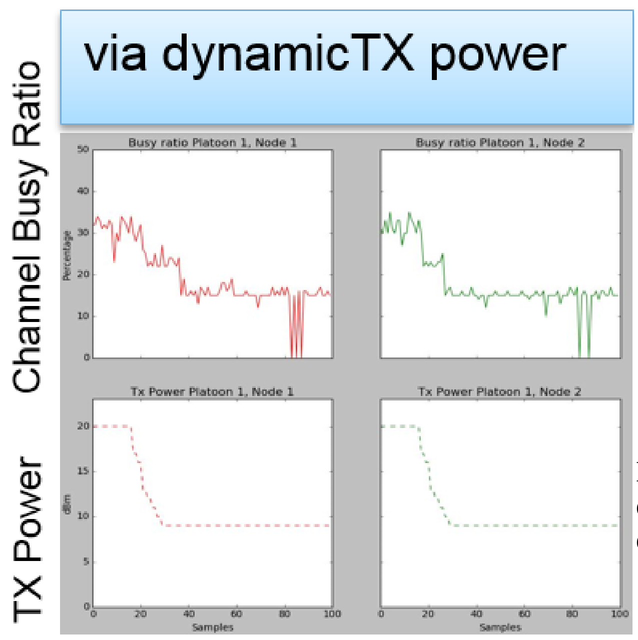
Methods for dynamically influencing the communication load (channel busy ratio)


WME
Congestion Detection
Congestion Control
Application Layer (smart message generation and forwarding)
Network Layer (smart rebroadcast routing)
MAC Layer (priority differentiation)
Physical Layer (channel sensing and measurement)
Channel
CCH
SCH

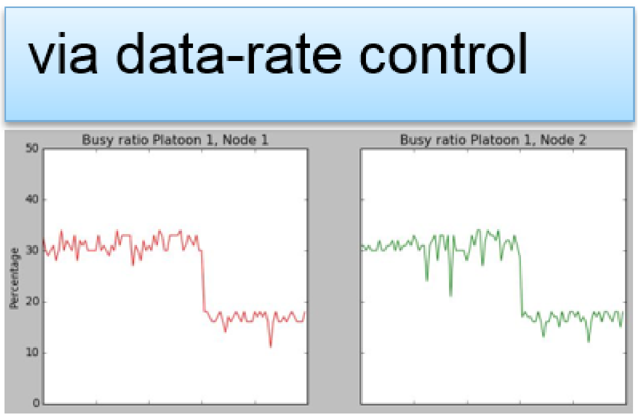
DCC methods can be implemented at various layers of the design

Figure 7: Decentralized Congestion Control applied to the NXP 802.11p wireless communication subsystem

An evaluation shows that 50% reductions in channel load are readily achievable.



Decentralized congestion control via congestion detection and dynamically adapting transmission power



Decentralized congestion control via congestion detection and dynamically adapting the applied data-rate scheme

Figure 8: Decentralized Congestion Control via dynamic transmission power and dynamic data control reduces channel busy ratio by 50%

5.1.2 Technolution Activities

Technolution has developed the core platform for the integration of each of the partners sub systems into a homogeneous whole. For this purpose modern small scale manufacturing techniques have been exploited to create a fleet of scale vehicles which can operate in a relatively small amount of floor space to demonstrate co-operating and autonomous vehicles in an urban or inter-urban environment. These vehicles have sensors that are representative analogues of those on production vehicles and the computing functionality is hosted either on the vehicle or in associated back-end systems. These vehicles have been used by both Denso and TUE for the development of their demonstration functionality.

During the project the vehicles and the sensor infrastructure on them have been iterated multiple times as more has been learned about the target environment and more competence has been developed. The current vehicles are based upon a NXP4370 multi-core CPU for low level control and a Raspberry Pi for high level control. A vehicle based upon this arrangement of components is shown in the figure below;

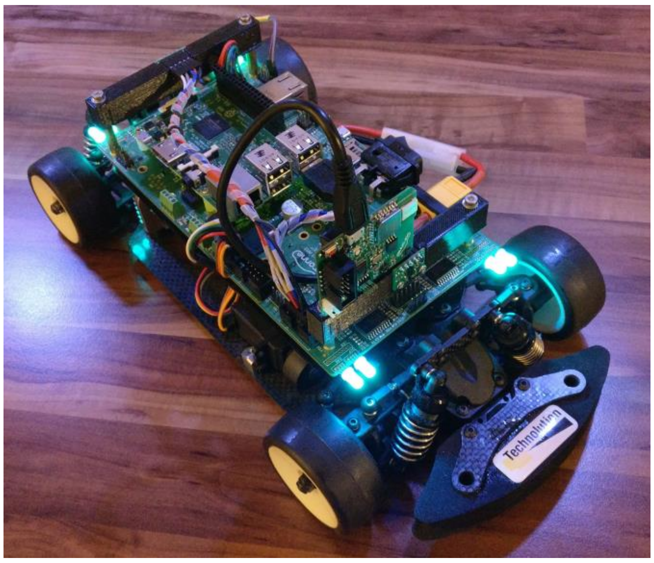


Figure 9: Technolution model vehicle

Alongside this a set of STM32L15x based boards have been developed that provide high speed communication, localisation and 9D IMU capability using an inertial navigation sensor and a Ultra-Wideband radio;

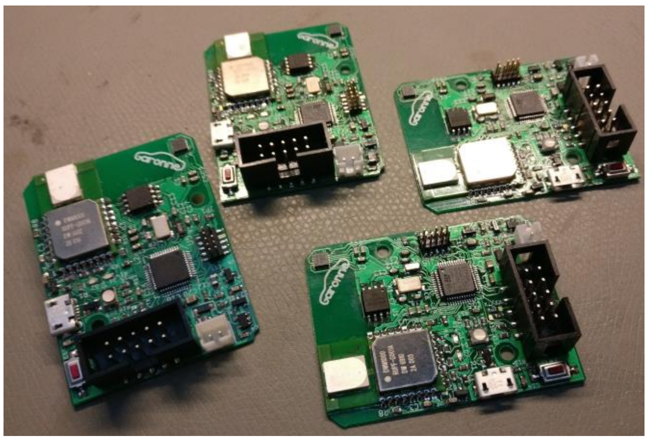


Figure 10: Prototype micro-location boards

One of these boards can be seen mounted on the front of the vehicle above. One set of three boards mounted at fixed locations provide a 3D localisation capability in the local environment. The boards also provide a V2I and V2V capability with a channel bandwidth of nearly 6Mbps. Characterisation results so far show a 3D positional accuracy of better than +/- 100mm in all axes.

Alongside this model system based work, Technolution has also developed a TEGRA-X1 based solution featuring 802.11p connectivity for use in the TomTom field testing;

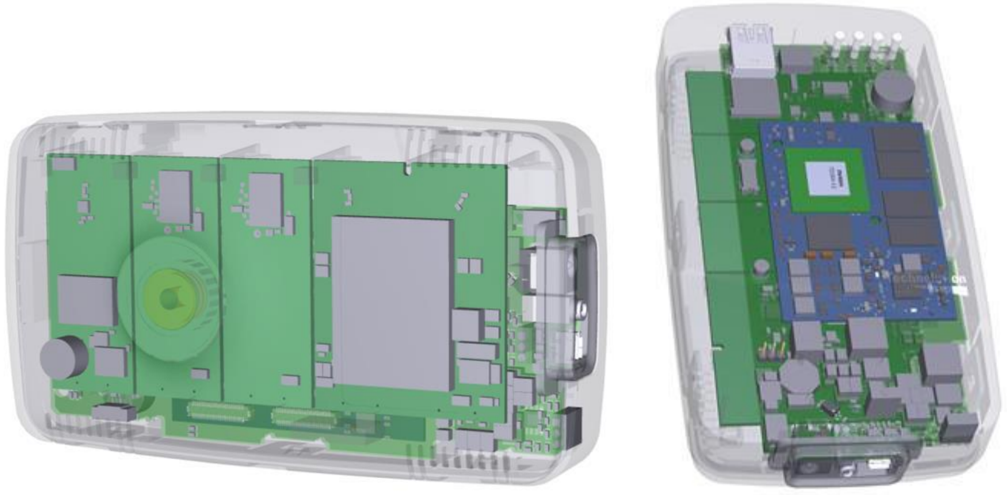


Figure 11: TEGRA-X1 integration for TomTom vehicle testing

More details about this unit can be seen in the TomTom section of this appendix 1.

5.1.3 DENSO Activities

DENSO has been working on a demonstrator which shows different technologies. The core technologies of this demonstrator are the migration of single core software to multi-core processors and the safe and dynamic update of functionality in an automotive environment. These technologies are demonstrated by a Green Light Optimized Speed Advisory (GLOSA) implemented on the model vehicles which were developed and provided by Technolution. Figure 12 shows the arena in which the model cars are operated. The arena consists of a closed course of approximately 4*5m² in which the model cars drive in a circle. The arena further consists of a traffic light. The cars are supposed to stop in front of a red traffic light. The traffic light sends Signal Phase and Timing (SPaT) messages to the model car which can used to optimize the speed of the car so that it passes the traffic light within its green phase if possible within the given contraints like maximum and minimum speed.

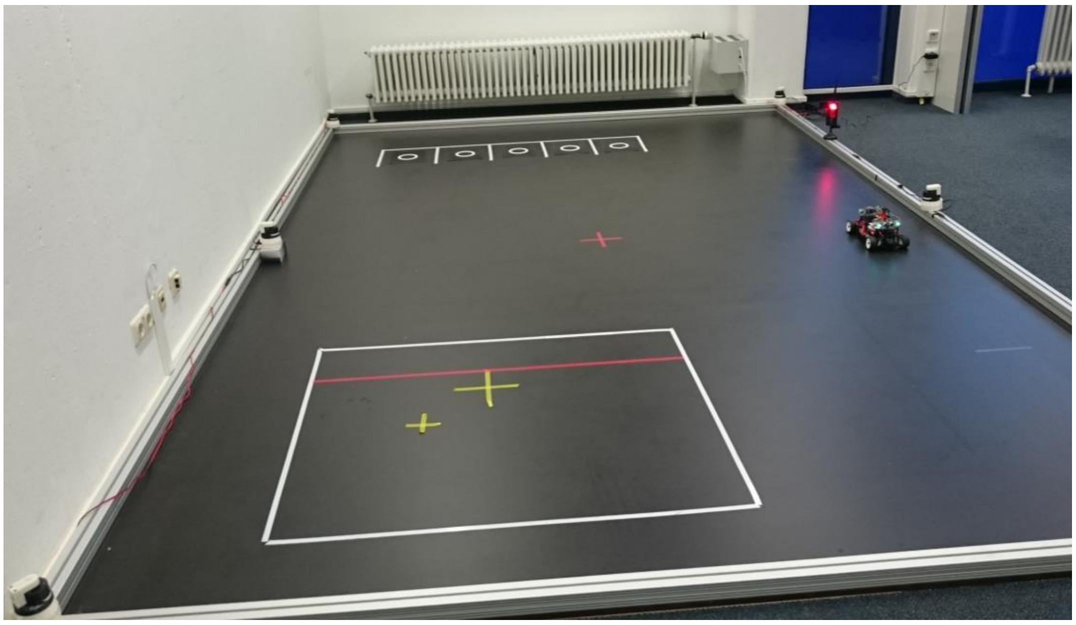


Figure 12: Arena for model cars

The demonstrator is build up from the modules as shown in Figure 13 below.

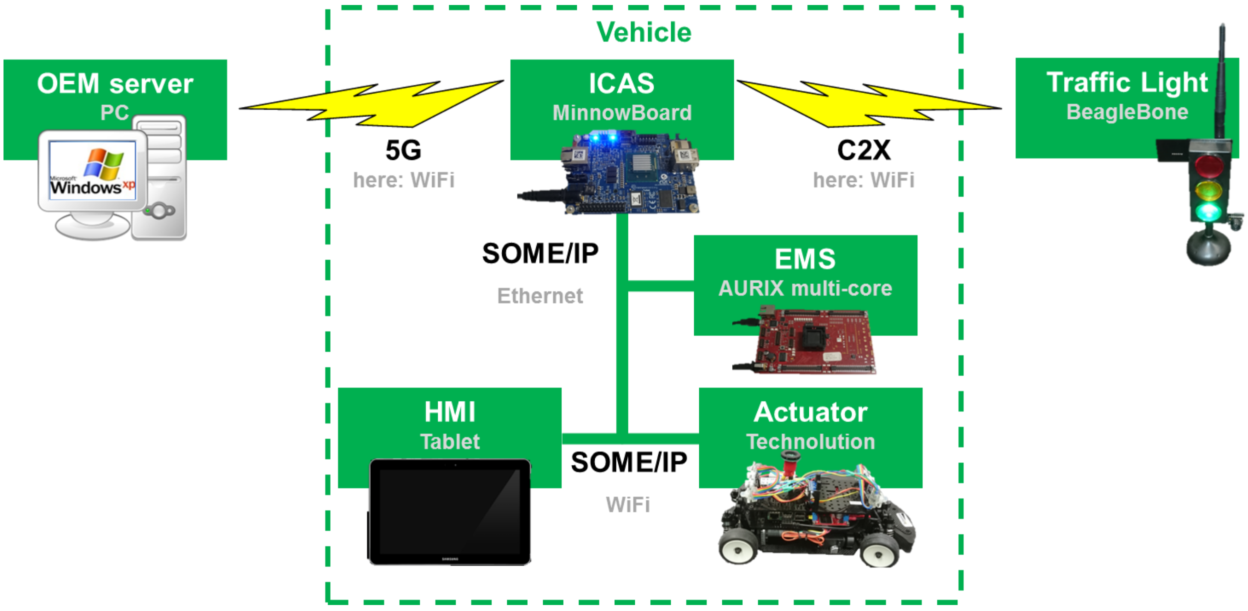


Figure 13: Structure of DENSO demonstrator

The demonstrator consists of three main modules which will be separated also later in the real world:

- OEM Server: A server managed by the car manufacturer which will announce and provide software updates over the air.
- Traffic Light: A traffic light which is C2X communication enabled. That means it can wirelessly transmit its signal phase and timing information
- Vehicle: The vehicle receives data from the OEM server and the traffic light, processes them and acts accordingly. Actions can be e.g. an update of the vehicle's software or a chance of the vehicle's speed. Within the vehicle there is an In Car Application Server (ICAS) which acts as a gateway into the car and central controller. For the demonstrator a Minnow Board is used as ICAS. Further the vehicle contains control software. Here as an example an engine management system has been chosen and ported to a multi core processor. For interacting with the driver the vehicle provides a Human Machine Interface (HMI) which is modelled in this demonstrator by a tablet. The last part are the actuators which take the input from the control software and realize them in the real world. Here the actuators are modelled by Technololution's model car. For flexible and dynamic communication structures within the vehicle the SOME/IP middleware is used.

In order to show the different aspects of the work done within this project the demonstrator shows different scenarios.
A central point of this demonstrator is the engine management system. The engine management system was migrated from a single core version to a multicore version. The parameters of the engine management system are shown in Figure 14. The first row shows the paramters of the multi-core processor like supply voltage, temperature, frequencies and core load. These paramters can be used to measure the efficiency of the migration method and to optimize the power consumption of the application. The second row shows input parameters for the engine management system like throttle and air mass. Here also parameters for the engine model can be set. The last row shows the output of the engine management system. This allows checking the correct function of this system.

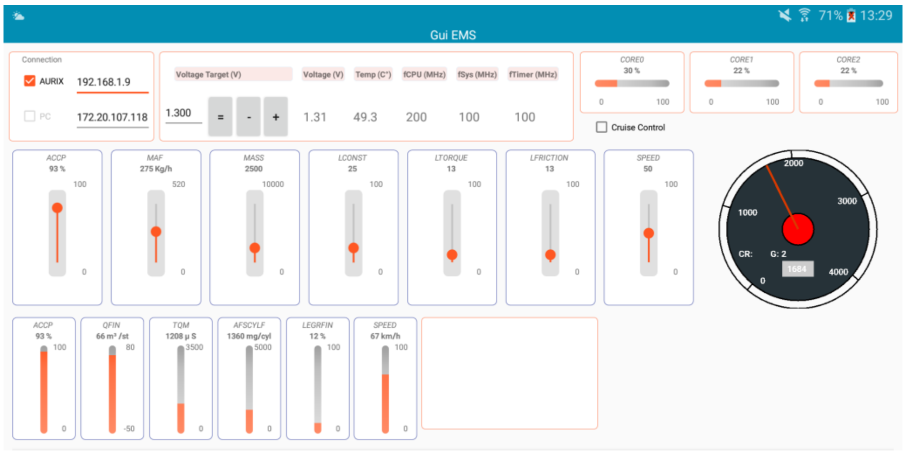


Figure 14: Engine Management System

The save and dynamic update is demonstrated by using two different software versions for an application running on the in-car application server. In the first version the application is only able to receive signal phase and timing (SPaT) messages from a traffic light and to display it on the HMI (Figure 15).

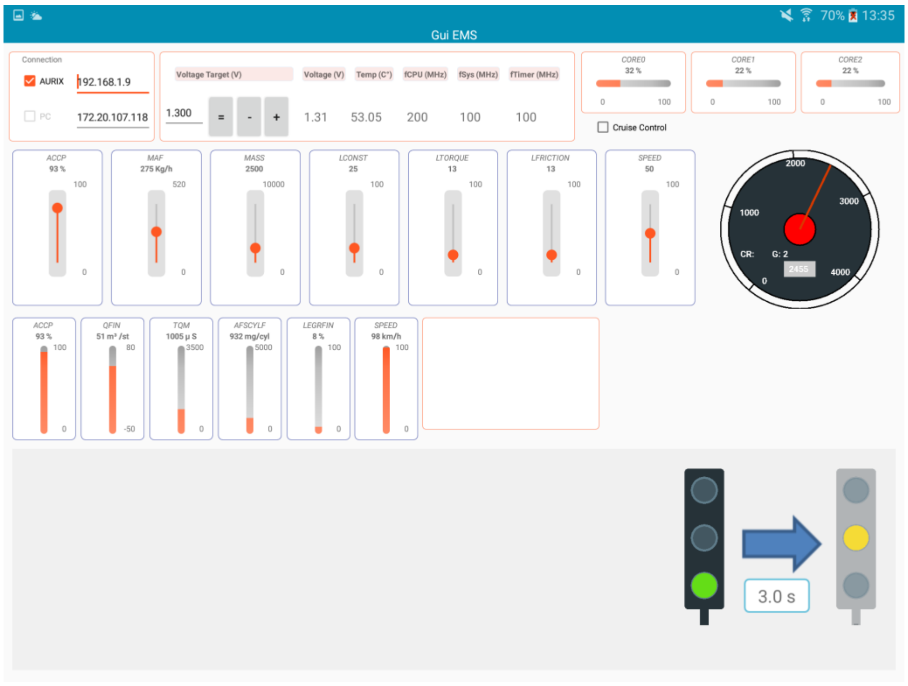
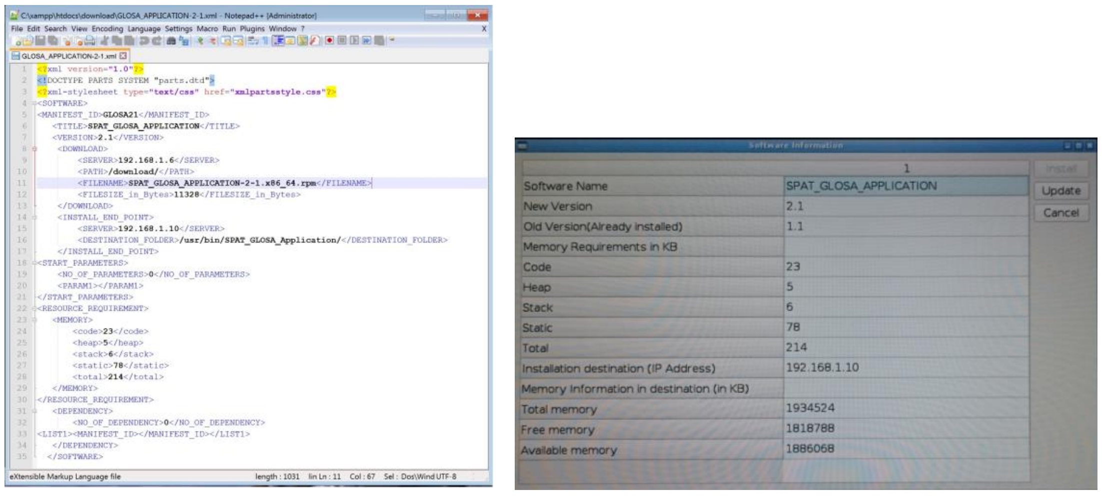


Figure 15: Displaying SPaT messages

For updating the SPaT software the OEM server informs the ICAS about a software upgrade (from SPaT to Green Light Optimum Speed Advisory (GLOSA)). The ICAS checks its available resources based on a software description (manifest). On the left side of **Figure 16** an example of such a manifest is given. For a safe and dynamic update it is essential that later when the new software gets operational no resource conflicts occur. Because of this the manifest includes information about the resource consumption by the new software. Here the memory consumption is indicated. Further the manifest contains information about dependencies on other software parts which may have to be installed on the ICAS. In this example the software has no dependencies. On the right side of **Figure 16** the GUI is shown which displays the content of the manifest and the available resources on the ICAS to make the checking process transparent for the user.



```
<?xml version="1.0"?>
<!DOCTYPE PARTS SYSTEM "parts.dtd">
<?xml-stylesheet type="text/css" href="xmlpartsstyle.css"?>
<SOFTWARE>
<MANIFEST_ID>GLOSA21</MANIFEST_ID>
   <TITLE>SPAT_GLOSA_APPLICATION</TITLE>
   <VERSION>2.1</VERSION>
    <DOWNLOAD>
        <SERVER>192.168.1.6</SERVER>
        <PATH>/download/</PATH>
        <FILENAME>SPAT_GLOSA_APPLICATION-2-1.x86_64.rpm</FILENAME>
        <FILESIZE_in_Bytes>11328</FILESIZE_in_Bytes>
    </DOWNLOAD>
    <INSTALL_END_POINT>
        <SERVER>192.168.1.10</SERVER>
        <DESTINATION_FOLDER>/usr/bin/SPAT_GLOSA_Application/</DESTINATION_FOLDER>
    </INSTALL_END_POINT>
<START_PARAMETERS>
    <NO_OF_PARAMETERS>0</NO_OF_PARAMETERS>
    <PARAM1></PARAM1>
</START_PARAMETERS>
<RESOURCE_REQUIREMENT>
   <MEMORY>
       <code>23</code>
       <heap>5</heap>
       <stack>6</stack>
       <static>78</static>
       <total>214</total>
   </MEMORY>
</RESOURCE_REQUIREMENT>
   <DEPENDENCY>
       <NO_OF_DEPENDENCY>0</NO_OF_DEPENDENCY>
<LIST1><MANIFEST_ID></MANIFEST_ID></LIST1>
   </DEPENDENCY>
  </SOFTWARE>
```

	1
Software Name	SPAT_GLOSA_APPLICATION
New Version	2.1
Old Version(Already installed)	1.1
Memory Requirements in KB	
Code	23
Heap	5
Stack	6
Static	78
Total	214
Installation destination (IP Address)	192.168.1.10
Memory Information in destination (in KB)	
Total memory	1934524
Free memory	1818788
Available memory	1886068

C:\xampp\htdocs\download\GLOSA_APPLICATION-2-1.xml - Notepad++ [Administrator]
GLOSA_APPLICATION-2-1.xml
eXtensible Markup Language file
length : 1031 lin Ln : 11 Col : 67 Sel : Dos\Wind UTF-8
Software Information
Install
Update
Cancel

Figure 16: Manifest and resource check

If the required resources are available, the ICAS asks the driver for permission to install the update (Figure 17).

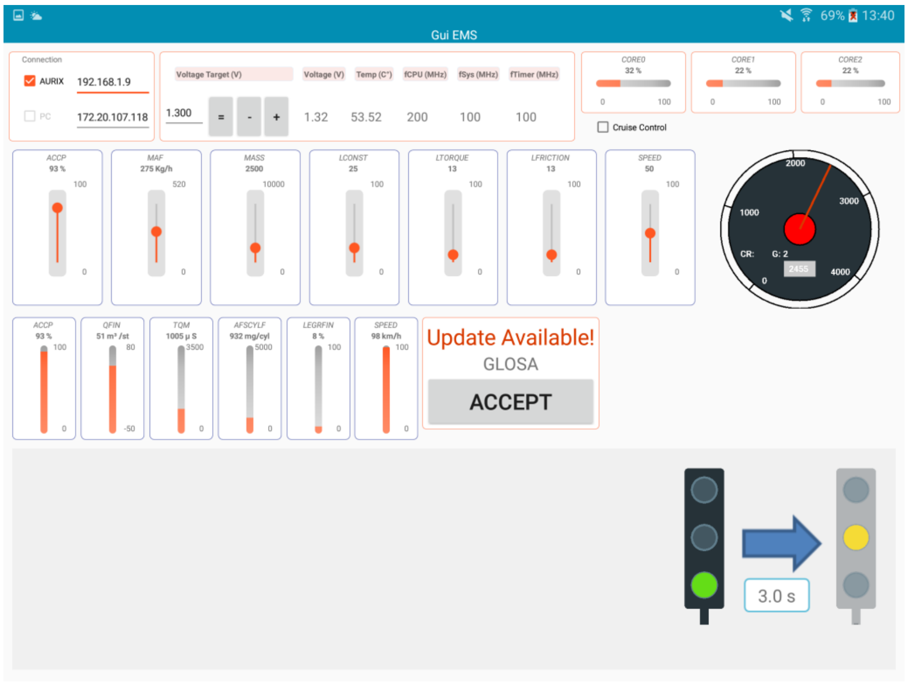


Figure 17: Update offered

After the software update the ICAS is now able to control the speed of the vehicle based on the received SPaT messages from the traffic light. . It reads the vehicle's current position and calculates the distance to the traffic light. Together with the SPaT messages this allows the ICAS to calculate and set an optimal speed for the EMS by using the EMS' cruise control feature. The communication of the required values between the ICAS and the EMS is done by using the middleware SOME/IP.

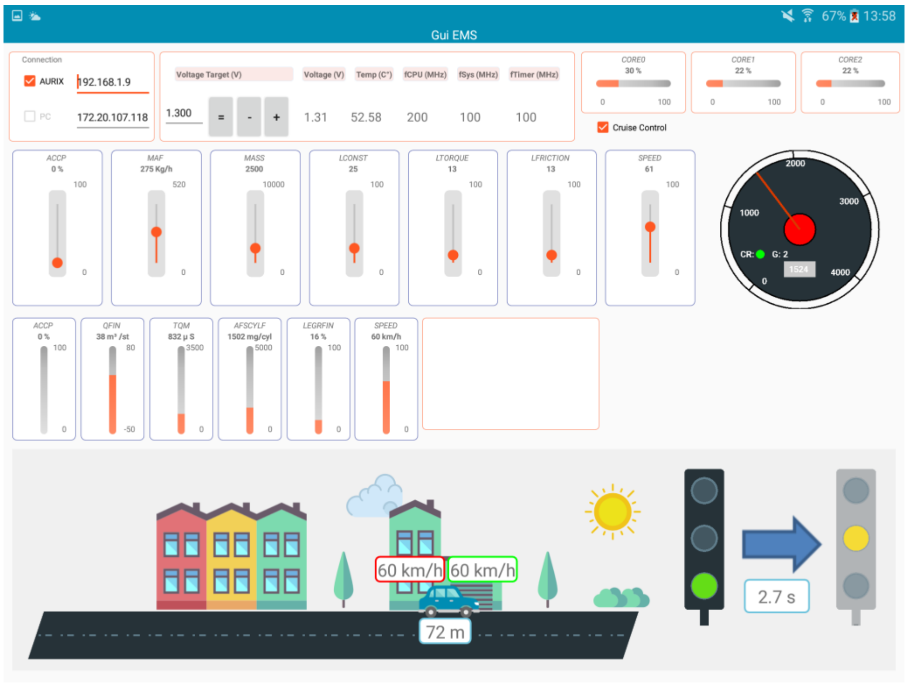


Figure 18: GLOSA

5.1.3.1 Performance Evaluation

Denso evaluated the parallel EMS implementation with regard to performance, efficiency, and potential for energy-saving. The case study is composed of more than one thousand runnables distributed among ten periodic tasks. An 11th task is trigger by crank-angle events.

The AURIX microcontroller runs the same EMS application during all measurements, which is parallelized to run on its three cores. The test application is parallelized with the runnable-level approach RunPar [4]. The AUTOSAR stack Elektrobit tresos AutoCore Generic (WP3) runs the application. Additionally, one of the cores runs an Ethernet stack, which is used to exchange data with a host PC. The data exchange is carried out to monitor the behaviour of the application, e.g. to check if all parameters are calculated correctly. Moreover, we set the voltage for the core logic (using the microcontroller embedded voltage regulator) and display the core voltage, which is measured by an internal ADC.

5.1.3.1.1 Performance and Efficiency

Denso evaluated the improvement by parallel execution and the overhead introduced by the operating system. The execution time was measured with a Lauterbach debugger. Table 8 shows the speed-up and the overhead for each periodic task in the use case. The parallel execution time is compared with the serial execution time on one of AURIX's performance cores. The checkmark (✓) means the speed-up is larger than 1.1 and thus the parallelization is successful. The exclamation mark (!) means the speed-up is between maximum 1.1 and thus there is small benefit from parallelization. The cross (✗) the speed-up is smaller than 1.0 and thus there is no benefit.

Task	1	2	4	8	16	20	32	64	96	128	1024
vs. Task	✗0,8	✗0,3	✓1,2	✓1,4	✓1,2	! 1,1	✓1,2	✓1,3	! 1,1	! 1,1	✓1,5
vs. Total	✗0,7	✗0,4	✓1,2	✓1,4	✓1,2	! 1,1	✓1,2	✓1,2	✗0,9	✗0,9	✓1,5
Barrier wait	73%	88%	40%	0%	19%	29%	5%	13%	23%	31%	0%
OS Overhead	68%	22%	5%	5%	14%	13%	5%	9%	14%	2%	13%

Table 8: Speed-up per task and OS overhead

The results show that tasks 1 and 2 do not benefit from parallelization. Both tasks spent most of their time in a barrier, waiting for the other core. Moreover, the OS overhead is large in comparison to the computation time of the task itself leading. Both lead to a speed-down. As a result, the task should be executed in a serial way instead.

Possible reasons for this speed-down are under investigation. One possibility is jitter between task activations, making one task start before other one. This leads to additional waiting time. Another possibility is a wrong estimated for the execution time of a runnable. The execution time is required to distribute the runnables with the runnable-level parallelization heuristic RunPar.

A better performance is achieved with tasks 20, 96, and 128. They provide a speed-up of 1.1. The overhead, however, leads to a speed-down of tasks 96 and 128. Thus, task 20 can be executed in parallel and tasks 96 and 128 should be executed in a serial way.

Contrarily, tasks 4, 8, 16, 32, 64, and 1024 benefit from parallelization. The speed-ups range from 1.2 to 1.5. Even when the OS overhead is taken into account, the performance is improved.

The system configuration was adjusted afterwards to improve the performance. The overall speed-up, considering the activation frequency for each task, is 1.2.

5.1.3.1.2 Energy-Saving

Denso investigated how the combination of runnable- and task-level parallelism can explore the aspired energy-saving potential of multicores under strict latency constraints. As a result, the work in [5] proposes the energy- and latency-aware parallelization approach Parcus. In this paper, Denso evaluated the potential for decreasing the processor's clock rate. Subsequently, they investigated how the observed clock rate reduction translates in energy-saving on a real processor.

Energy-saving on the Infineon AURIX

The energy-saving depends on the processor architecture. The Infineon AURIX TC277 with three cores is considered; a typical representative for an automotive embedded multicore processor. This microcontroller is fabricated in a 65nm technology. Its core logic is qualified for an operating voltage of 1.3V and the maximum clock rate is 200 MHz.

Denso measured the lowest supply voltage at a predefined frequency, for which the core logic is still operational. Therefore, the voltage is lowered until the microprocessor does not operate in a normal manner any more. Only the internal flash and SRAM are used as memories. The bus/NoC frequencies are scaled according to the core frequencies. Hence, the full system scales linearly. Criteria for normal operation are correct results and a working connection to the PC via Ethernet. The microcontroller runs for several seconds on each voltage level. In cases of an insufficient voltage setting, the microcontroller stops working at once. Thus, stable operation can be assumed after several seconds of operation. We are well aware that the microprocessor is operated out of its specification and hence the failure rate is higher under these conditions. For all frequencies, the same relative safety margin of 30% for the voltage is used; as observed at 200 MHz. The lowest observed working voltage here is 1.0V, whereas the specification demands 1.3V.

The measured lowest working voltages are listed in column 2 of Table 9, for the frequency listed in column 1. The resulting supply voltage with safety margin is listed in column 3. We assume these values for further experiments. Column 4 lists the energy per operation relative to the 1.3V operating point. The energy per operation is proportional to the square of the supply voltage. The results in table I show that lowering the frequency from 200MHz to 100MHz allows reducing the supply voltage for the core logic by about 14%. This reduction of the supply voltage leads to 26% less energy per operation. However, further reducing the frequency, i.e. below 100MHz, does not give any additional headroom to further lower the core logic supply voltage and the energy per operation. Hence, the potential of energy-saving by voltage-frequency scaling is limited for this microcontroller to 26%.

Frequency	Voltage	Voltage + margin	Relative energy
200 MHz	1.00 V	1.30 V	*1.00*
175 MHz	0.96 V	1.25 V	*0.92*
150 MHz	0.93 V	1.21 V	*0.86*
120 MHz	0.88 V	1.12 V	*0.77*
100 MHz	0.86 V	1.12 V	*0.74*
75 MHz	0.86 V	1.12 V	*0.74*
50 MHz	0.86 V	1.12 V	*0.74*

Table 9: Measured lowest working core voltage depending on frequency.

Figure 19 shows the cubic Hermite spline interpolation of the voltage-frequency scaling of the AURIX based on the measurements conducted in the experiment before, listed in table I. This curve is used to determine the energy-saving.

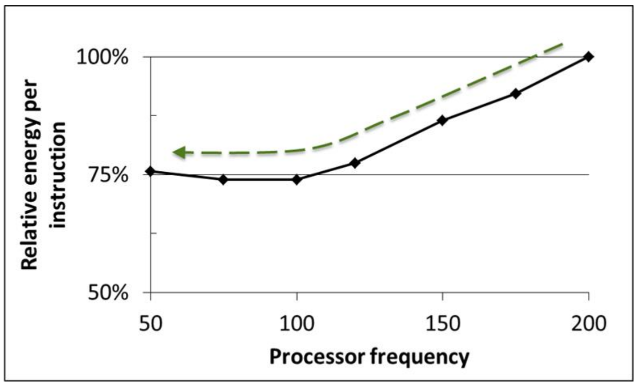


Figure 19: Interpolation of the relative energy consumption per instruction on the Infineon AURIX.

Energy-saving through Combination of Runnable-and Task-level Parallelism

Figure 20 shows the resulting curves that are named SCT (single-core task), RunPar (parallel task), and SCT+RunPar, respectively. The energy-saving is equal when 18% to 41% of the inter-task communication is replaced by *timed implicit communication* (TIC). In this area, all combinations of runnable- and task-level parallelism fully utilize the available energy-saving margin of the AURIX. Replacing more inter-task communication by TIC (i.e. introduce more task-level parallelism) shrinks the energy-saving only in the case of RunPar, because the adjustment for robustness increases the latency. In contras, only relying in task-level parallelism (SCT) only provides minor energy-saving potential when applied rarely, because most of the tasks need to be executed in sequential order.

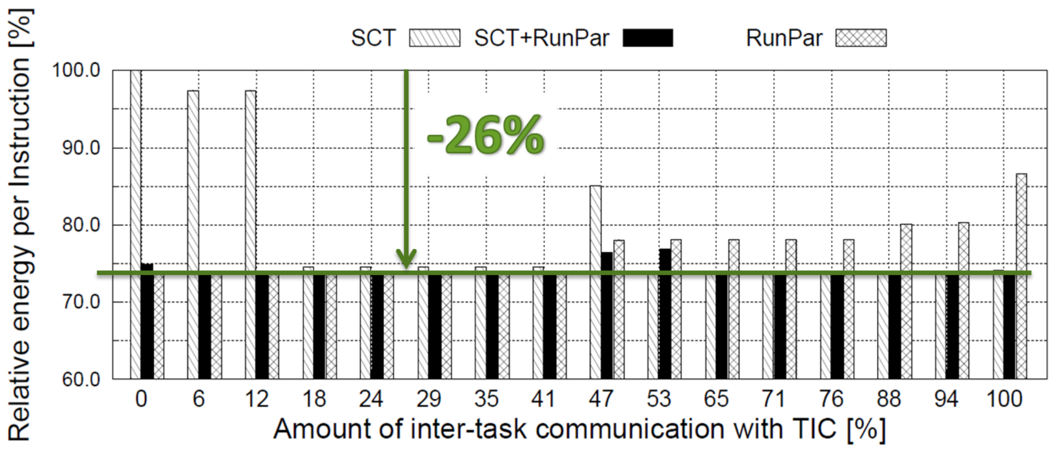


Figure 20: Relative energy consumption per instruction.

5.1.4 TUE Activities

TUE using CompSOC platform (compsoc.eu) which allows for synthesis of tile-based architecture configuration with multi-processors (processor tiles), interconnections (Network-on-Chip (NoC)) and memories (memory tiles) within the same platform. Each processor tile is mainly composed of a MicroBlaze softcore processor. The monitor tile is specialized for debugging purposes. The memory tile contains the external memory interface and controller, and the NoC provides interconnection between the tiles. To enable independent implementation, verification and execution of multiple applications of different criticality (safety-critical, real-time and non-real-time), the CompSOC offers composability and predictability by virtualizing all processors, interconnections, and memory resources. An example template is shown below. Using the above tile-based architectute, the following functionalities are investigated;

- TUE is investigating the possibility of handling failure or fault-tolerance in the multi core mixed-criticality application architecture CompSOC. The idea is to use the resource (re)allocation strategies in the case of a failure of a CPU subsystem/tile on the in-vehicle platform (i.e., CompSOC in this case) taking into account high-level application requirements such as quality of control (QoC) of a feedback control loop. For this purpose, TUE has developed software architecture to dynamically create and manage multiprocessor partitions. This is done by a method for composable dynamic loading in uniprocessor and multiprocessor platforms which ensures that loading applications do not affect the running applications and vice versa. Furthermore the loading time is also predictable i.e. the loading time can be bounded a priori. This is achieved by splitting the loading process into parts, wherein only a small part which reserves minimum required resources is executed in the system partition and the other parts are executed in the allocated application partitions which ensure isolation from other applications. Current partitioned systems do not allow dynamically adding applications. Applications are statically loaded in their respective partitions.
- Further study will address aspects such as timing constraints on dynamic reloading imposed by application-level requirements (e.g., QoC, throughput) and software, hardware requirements or resource allocation strategies for meeting such constraints coming from the applications.

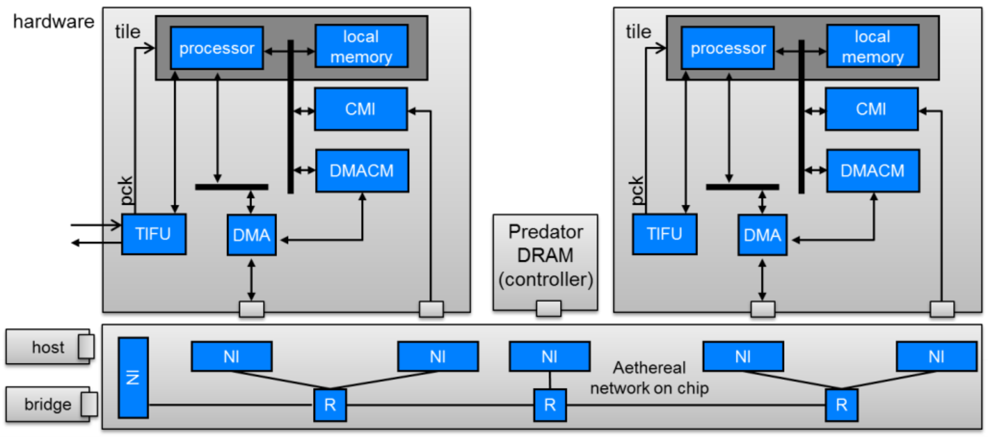


Figure 21: Example parallel CompSOC implementation

Evaluation results

TUE has integrated the CompSOC platform with the EMC² car (from Technolution). The CompSOC platform implements the function of high-level controller where it, based on sensory information from the Low Level Board (from Technolution), takes decisions and send driving instructions back. As a second function it visualizes the current state of the car on a screen. The demonstration, see figure attached, shows the Xilinx ML-605 prototype board running a 3 processor instance of the CompSOC platform that is connected to the EMC² car Low Level Board via a serial connection and a monitor for visualization. In the experiment, the car driving into a dead-end ally, the high-level control detecting that it hit a dead-end and giving the commands for making a three point turn to drive out of it again.

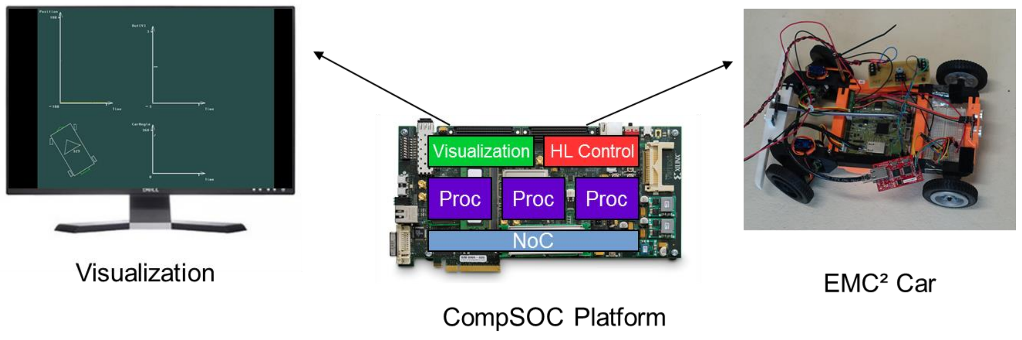


Figure 22: Overview of the integrated setup

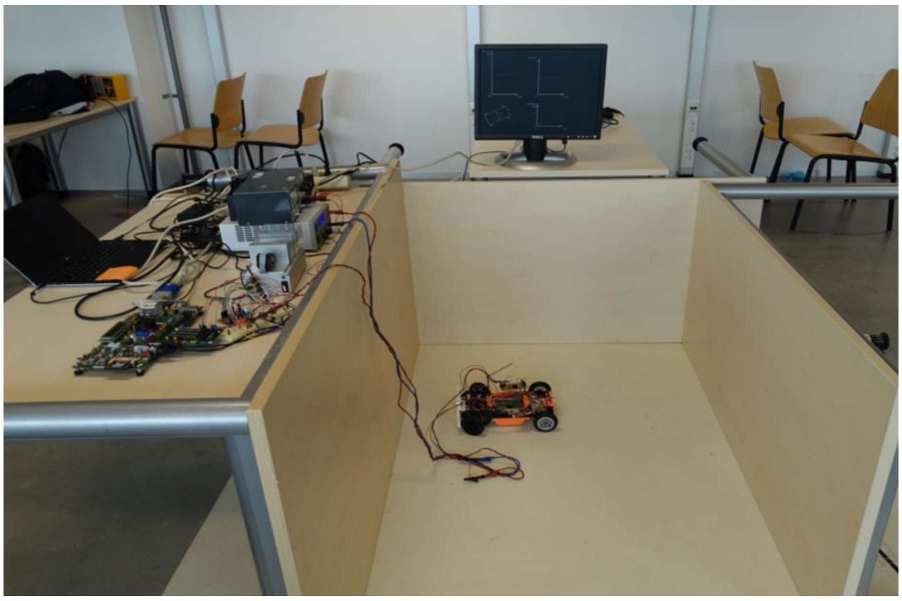


Figure 23: Snapshop of the experiments

- KPI29 number of control units: The integrated setup based on CompSOC platform allows for reducing the requirement on the number of dedicated processing units
- KPI32 efficiency of control algorithms: Due to highly parallel and reconfigurable architecture, the control application can adapt the compute requirement based the system dynamics making it efficient.

5.1.5 TomTom Activities

Embedded Platforms for Lane Level Navigation

Several embedded platforms were created for in-car experiments to run the 'dynamic lane level navigation' functionality. The following product concepts were created:

Table 10: Overview of Lane level navigation platform prototypes

Platforms	Section	Map	Computation	Sensors
Off-the-shelf	2.1	Regular	Qualcomm Snapdragon	Consumer-grade
Retrofit prototype	2.2	Enhanced	NVidia Tegra X1	Automotive-grade
In-car prototype	2.3	Enhanced	DrivePX2 or similar	Automotive-grade

An off-the-shelf navigation device was used with an intergrated camera, that was extended with software detecting lane markings and matching the position with the observed lane markings to a lane position in the map. The navigation application was enhanced with a widget in the HMI display, showing the lane the car is driving in. For this product concept, the standard map was used.

Evaluation result:
This approach using off-the-shelf navigation devices is not expected to lead to a viable product, for reason that the fusion of consumer-grade GPS, current navigation maps and camera in an after- market device does not lead to the required lane positioning accuracy to be able to create a good user-experience.

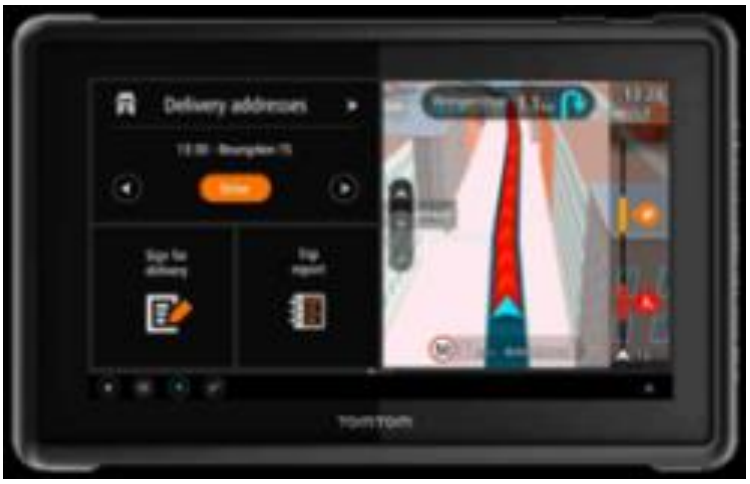


Figure 24: Off-the-shelf prototype

In a follow-up step the hardware configuration was changed to a navigation device based on astate-of-the-art COTS SoC with a powerful multi-core CPU and GPU (Android tablet) to run the dynamic lane navigation functionality and an accessory to run the lane positioning functionality with automotive grade sensors. The accessory was created by Technolution upgrading their V2X platform with a computer board with a multicore GPU that can accelerate the perception software, with an Automotive grade camera and a more accurate GPS/IMU technology.

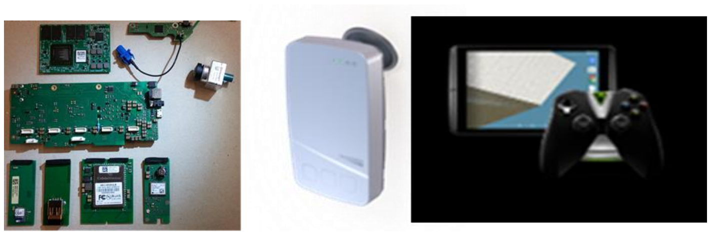


Figure 25: Retrofit prototype

Evaluation result:
This hardware for this setup is just avialable and is being integrated in a R&D car build for field trial experiments. Coming monthes field tests are planned in. The test results are not yet available.

HD Maps (TomTom)
Maps used in the EMC^2 project are based on NDS. NDS is a standardized format for automotive-grade navigation databases, jointly developed by automobile manufacturers and suppliers. The current NDS production maps are originally designed for road-level navigation systems. To make these maps suited for lane level navigation functionality, the NDS specification map is extended with extra attributes for precise localization and lane level navigation. The extra attributes are organized in NDS as extra layers stacked onto the maps currently used in production maps (SD maps), see also figure below.

Evaluation result:

HD-map samples were created for a test track in Eindhoven and succesfull in-car dynamic-lane-navigation test drives were driven with very good feedback on the experience of lane guidance functionality.

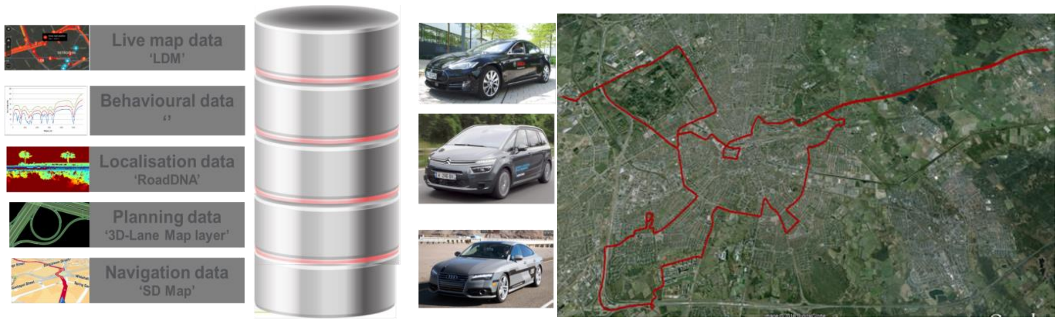


Figure 26: TomTom HD maps layers and coverage of TomTom HD maps in the Eindhoven test area.

Map making based on crowdsourcing in-car sensor data

One of the concept ideas tried out in EMC^2 was to research the technical feasibility to create maps from camera sensor data sourced from consumer devices. For that the experiment was set-up to update speed limit information in the map on the basis of speed limit signs recognized in the car together with (GPS based) location data. In the cloud several in-car observations were aggregated into evidence that a map update can be made from the observation data. For that perception and classification algorithms were developed and a platform to process observations in the back-office into map-updates that were communicated back to the device in the car. In a later stage the same principle was applied to the mining of lane markings.

Evaluation result:
The principle works out for maps used in infotainment systems, but will not work out for maps used for ADAS and AD driving systems. For reason that the quality of the observation data from consumer grade sensors that are not well calibrated is not sufficient to build HD-maps that require higher accuracy than for navigation maps.

Dynamic Lane navigation with SDK for integration
To be able to accommodate developments and experiments on all platforms the HMI implementation within the navigation application is based on a QT platform that is derived from a commercial navigation application of TomTom. This platform is adapted to connect to the Lane Navigation engine developed for EMC^2 and to add featuring necessary to do specific experiments as defined in EMC^2, such as the use of dynamic information in the navigation functionality.

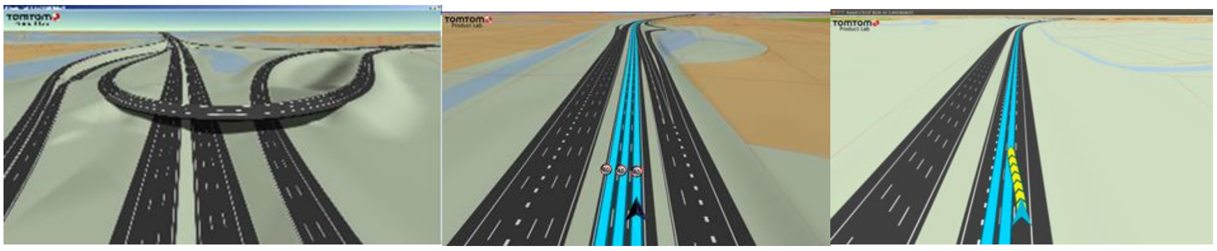


Figure 27: Output of the 3D lane renderer.

Evaluation result:
A full functional lane navigation application and SDK is available with the intention to re-use this application and tools in other partner research programs as an open navigation platform to implement cooperative ITS and autonomous driving use cases.

5.1.6 TNO Activities

In order to reduce the number of hours required to test an ADAS system (KPI_T7.1_01), TNO adopted scenario classification as one of the use cases. A scenario database contains safety critical situations that appear during real world driving. By parameterizing relevant scenarios by means of probability distributions (which are learned from real world data) the majority of performance evaluation and ADAS development can be done in realistic simulations and the amount of driven kilometers during the ADAS verification phase reduces drastically. During the last year, TNO turned the offline scenario classification algorithm into an online algorithm that runs while driving. The classification is now performed in two steps. In the first step, a so-called preclassifier, removes trajectories based on rules. For example, if an object only appears for a few time steps, it does not pass the preclassifier. The second step is the actual classifier, based on a model which is trained offline using supervised data of car-cyclist scenarios. The accuracy of the model is increased with respect to last year by using additional data gathered throughout the year.

In order to reduce the number of platforms in a vehicle, the scenario classification application runs on a single multicore platform together with the world model and a cyclist AEB application (KPI_T7.1_04). This is enabled by adopting a safety-executive software architecture that keeps track of the correct working of the overall system. More information about this architecture is given in sub task description ST7.1.3 later in this appendix.

TNO implemented a safety-executive architecture to ensure safe operation and redundancy at a software level. This architecture is particularly useful for a single embedded platform running multiple mixed-criticality applications in parallel. A health monitor continuously monitors the system's health and whenever a sensor or (sub)system fails, the health monitor switches from the nominal mode to a safe mode only executing the minimal required (safety critical) software in a safe and predictable manner. The overall system will be demonstrated during the final review event by means of a video.

As part of the "thinking" component, on top of the real time sensor layer developed last year (part of the "sensing component") a world model has been developed. The current world model takes lidar data, tracks bicycles and compensates for the ego motions of the vehicle. It predicts motions for objects that are temporarily occluded and may exploit road information, e.g., lines detected by a camera, to discard irrelevant portions of lidar data. In order to enable redundancy in software facilitated by the safety-executive pattern a second highly predictable (but less advanced) real-time world model has been implemented based on a single radar sensor. This system has lower complexity, is more predictable and therefore is a safe backup in case of failures. As a result, the overall system is able to handle failures in, e.g., the lidar or the multisensor based world model.
Finally, at TNO, two applications ("act" component) were developed for the purpose of testing all elements in the sense-think-act chain: a very simple cyclist AEB algorithm and an advanced scenario filtering algorithm.
For safety reasons, the cyclist AEB does not actuate the vehicle in real world driving but gives a visual indication instead. The cyclist AEB system is the application with highest priority since it is clearly safety critical and may not fail whatever happens with the other application. It is able to run in both the nominal and the safe mode, i.e., using either the full or the simplified world model. The scenario classification algorithm runs permanently but is not safety critical for the driver. It will therefore only run whenever the overall system works the way it should (as is monitored by the health monitor). More details about the algorithm, the purpose of the algorithm and the advancements last year were given in the KPI section at the beginning of this appendix.

Set-up and execute trials
For the real world trials, TNO has used a Toyota Prius equipped with various sensors among which lidars, radars, cameras and on-board vehicle sensors, e.g., measuring wheel speeds. A real time sensor middleware has been developed that enables reading out sensors with minimal jitter and a world model fuses the information from all sensors as explained before. A picture taken during one of trials showing the test vehicle is shown below. The focus during the field trials was on car-cyclist scenarios since accident statistics prove the importance of these scenarios.

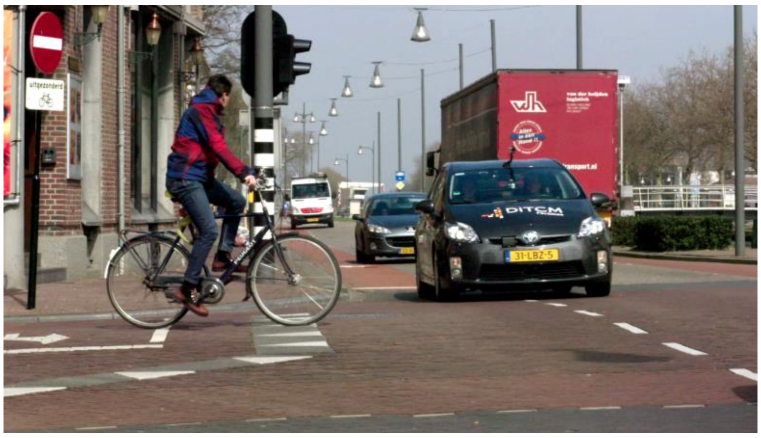


Figure 28: Typical detection scenario

As indicated in different parts of this appendix, TNO has demonstrated how two different applications with a different criticality can run in a safety executive architecture on a single multicore platform. The platform is integrated in a real vehicle and the applications are validated in real traffic.

This task contributes to the overall D7.2 deliverable and represents the assessment of the contribution of the work.

5.2 Exploitation

Technolution has used the techniques and technologies developed in this project for the WePods project for on the road autonomous vehicle driving as previously disclosed in D7.4. The Multi-core CPU board developed has been re-used extensively on other projects (along with the multi-core software competence required to exploit it) and the positioning system is in the early stages of commercialisation. Denso, Tomtom and NXP are folding their results into their normal lines of business. TUe continues to develop the parallel resource partitioned computing concepts within their active research teams.

6. Appendix 2: T7.2 UC Highly automated driving

6.1 KPIs

A set of KPIs hass been formulated based on the Business Needs (Appendix 2, 6.3) associated with T7.2. The table below shows the KPIs mapping the business needs

Table 11: KPIs mapping the business needs

No.	Title	Description	BN relation
KPI_T7.2_01	Increased coverage and configurability by simulation	Reduce cost and time for each system design (or adaptation) by at least 10%, while increasing 10% performance and energy efficiency.	BN_T7.2_01
KPI_T7.2_02	Decrease development time by harmonization of development tools	Reduce development time of 10% with respect to the current tool boundaries.	BN_T7.2_02
KPI_T7.2_03	Definition of methodologies to increase confidence in HW/SW integration	Defined methodologies/guidelines to efficiently integrate existing high-level functionalities under a mixed-criticality environment in multicore platforms	BN_T7.2_03
KPI_T7.2_04	Enhance the resources allocation	Increase by at least 20% the level of performances of the different sub-systems by a time-critical dynamic allocation mechanism.	BN_T7.2_04
KPI_T7.2_05	Definition of methodologies to characterize effective portability of SW	Defined methodologies/guidelines to characterize the suitability, constraints, flexibility and reusability of SW for specific HW configurations.	BN_T7.2_05
KPI_T7.2_06	Definition of tools to parallelize and classify SW susceptible to be accelerated	Defined concepts/methods/tools to semi-automatically determine which components or algorithms are susceptible to be accelerated and quantify the resultant performance gains.	BN_T7.2_06

6.2 Sub task descriptions

The figure below shows the tasks within this UC and the relation of each task both with other tasks within this UC and with other technical WPs. As can be seen in the figure, subtasks ST7.2.3 – ST7.2.5 are rather independent and mastered by subtask ST7.2.3. These four tasks are expected to be carried out following the requirements and evaluation plans defined in subtask ST7.2.1, and thereafter integrated and evaluated in ST7.2.6. The scope and more specific description of the UC are defined within this task.

Requirements are derived and collected in ST7.2.1 along with definitions of use case specifications, evaluation criteria and evaluation methodology. The requirements are communicated to the relevant tasks of this UC, Automotive Living Lab (WP7), and the other technical WPs (WP1, WP2, WP3, WP5, WP6) to perform the required technical activities.

General concepts, techniques and tools are developed in the respective technical WPs. Concepts, methods and tools developed in other WPs are to be customized to meet the requirements and specifications of the UC. Note also that since there are many links among most of the Automotive Living Lab Use Cases, synergies will be exploited to maximize the workflow between them. In particular, UC 2 and UC 1 share many functional requirements, so a strong collaboration between them will be pursued.

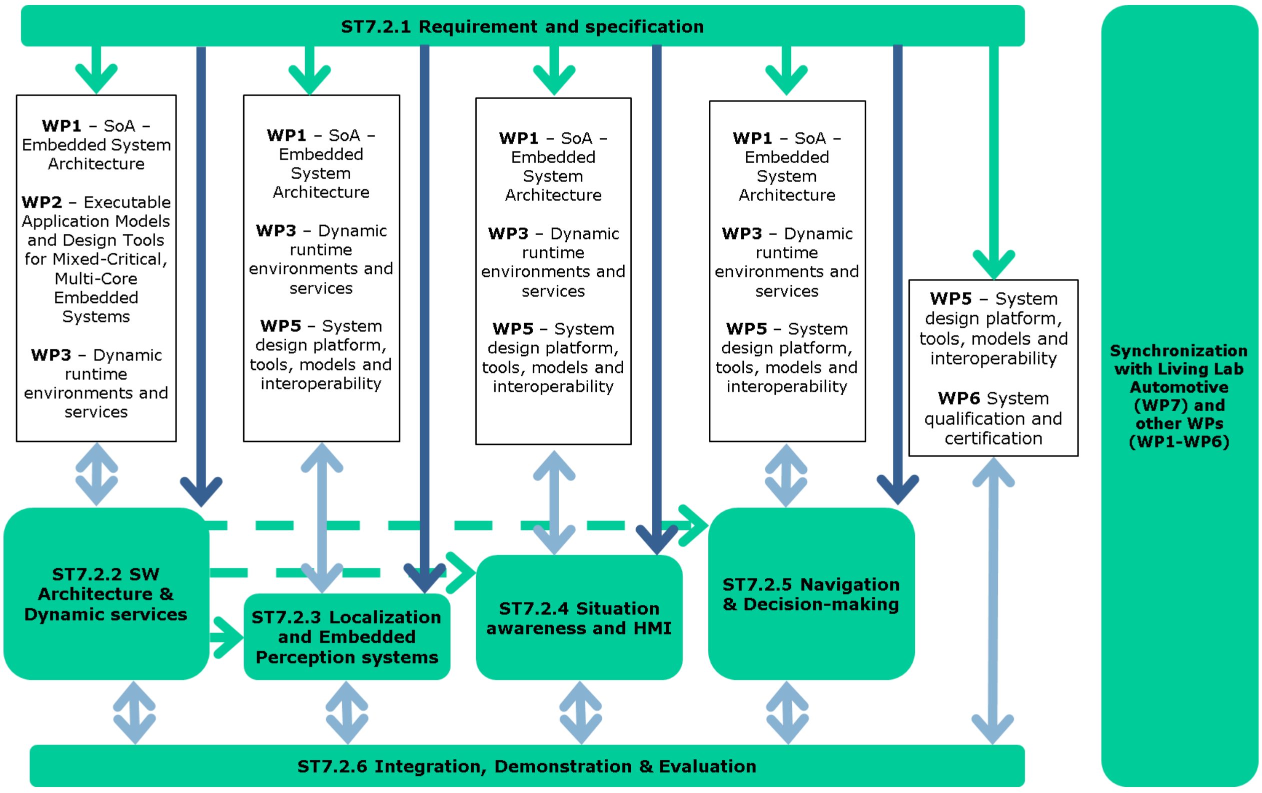

ST7.2.1 Requirement and specification
WP1 – SoA – Embedded System Architecture
WP2 – Executable Application Models and Design Tools for Mixed-Critical, Multi-Core Embedded Systems
WP3 – Dynamic runtime environments and services
WP1 – SoA – Embedded System Architecture
WP3 – Dynamic runtime environments and services
WP5 – System design platform, tools, models and interoperability
WP1 – SoA – Embedded System Architecture
WP3 – Dynamic runtime environments and services
WP5 – System design platform, tools, models and interoperability
WP1 – SoA – Embedded System Architecture
WP3 – Dynamic runtime environments and services
WP5 – System design platform, tools, models and interoperability
WP5 – System design platform, tools, models and interoperability
WP6 System qualification and certification
Synchronization with Living Lab Automotive (WP7) and other WPs (WP1-WP6)
ST7.2.2 SW Architecture & Dynamic services
ST7.2.3 Localization and Embedded Perception systems
ST7.2.4 Situation awareness and HMI
ST7.2.5 Navigation & Decision-making
ST7.2.6 Integration, Demonstration & Evaluation

Figure 29: Subtasks of T7.2 and interaction with WP7 and technical WPs (1-6)

ST7.2.1 Requirement and specification

The expected activities within this sub-task are as follows:

- Low level requirements and evaluation plans are derived from a first definition of business needs and key performance indicators.
- Specification of a workflow to properly integrate developments or tools provided by the partners of UC2, or other related UC and WP. It will be mainly based on an iterative development from concept definition to validation.
- Identification of the available or in progress tools and methodologies within the consortium that fulfill the workflow requirements. This task should permit to identify simulation, modelling and validation tools, HW specification, and SW development methodologies.
- Description of the UC goals, the demonstrators to be implemented, and the corresponding evaluation plans. The validation phase will be accomplished according to a set of criteria and metrics also defined in the low-level requirement specification.
- Subsystems technical specifications to properly coordinate the work between UC partners and other relevant partners from the automotive Living Lab and the technical WPs.

This sub-task contributes to the common WP7 deliverable (D7.1), and common T7.2 deliverables (D7.5 and D7.6)

Relation to business needs: BN_WP7_T7.2_01
Relation to WPs: WP1 and other Use Cases of the Automotive Living Labs

ST7.2.2 Architecture & Dynamic services

In this sub-task, a dynamic mixed-criticality on board system architecture is defined in order to guarantee high availability and reliability, taking into considerations local and remote sensors, processing and actuation elements. This task also uses modelling and simulation tools (WP2-WP5) for efficient HW/SW co-design of the embedded multi-core system. Dynamic scheduling approaches are evaluated in order to balance performance, flexibility and resource efficiency. The specific activities performed within this subtask are the following:

- Model behaviours of hierarchical, compositional, and executable SW components within EMC^2 SOA architecture in a multi-core target.
- Explicitly consider separation aspects among applications, with different criticalities and management of shared resources.
- Investigate a solution with an increased integration of both the existing functionalities and the improvement/refinements to be developed.
- Explore different strategies for dynamic scheduling and synchronization across the hybrid multi-core computing platform.

This sub-task contributes to the common WP7 deliverable (D7.2), common T7.2 deliverables (D7.5 and D7.6) and an internal report related to the SW Architecture.

Relation to business needs: BN_WP7_T7.2_01, BN_WP7_T7.2_02, BN_WP7_T7.2_04
Relation to WPs: WP1- WP3 and other Use Cases of the Automotive Living Labs

ST7.2.3 Localization and Embedded Perception systems

This sub-task involves the study and the development of the localization and standalone/cooperative perception systems embedded in the highly automated vehicle. Such systems will use Commercial Off-the-Shelf (COTS) sensors taking into account automotive integration constraints. The solution developed will explore the best trade-off performance/computational cost for the specified HW target. More specifically, the following activities are performed:

- Develop solutions that fuse information from the vehicle proprioceptive CAN-bus automotive sensors, geometrical features (road curvature and width) extracted by low-cost perception sensors and standalone low cost automotive-type GNSS receivers and Inertial Measurement Units;
- Implement reliable systems that perform large-field of view perception with COTS cameras and lidars/radars, being the detected objects static and linked to the infrastructure (traffic lights and signs) or moving (vehicles, pedestrians) on the drivable space. An investigation on the number, configuration and nature of these sensors has been performed to determine that the most suitable architecture for the selected HW target on the EMC^2 SW architecture basis is the fusion with multilayer lidar and stereovision data.
- Specify the percerption map characteristics and its attributes to properly use it for situation awareness purposes in the embedded solution.

This sub-task contributes to the common WP7 deliverable (D7.2), common T7.2 deliverables (D7.5 and D7.6) and an internal report related to the Localization and Embedded Perception systems.

Relation to business needs: BN_WP7_T7.2_02, BN_WP7_T7.2_05, BN_WP7_T7.2_06

Relation to WPs: WP1. WP3, WP5 and other Use Cases of the Automotive Living Labs

ST7.2.4 Situation awareness and HMI

In this sub-task, an overall dynamic local map will be generated from the onboard perceived entities and those received from the V2X communications. It will be the basis to derive a risk assessment considering estimated intentions and expectations of other road users, and for further decision-making. In the final step of the information flow, the driving situation interpretation will be properly shown to the driver in order to increase its situation awareness in case of take-over. This objective can be decomposed in the following activities

- Evaluation of resources allocation mechanisms for the generation of a Probabilistic Dynamic Local Map where several graceful degradation modes are to be considered.
- Exploitation of parallelization possibilities for risk assessment approaches based on Bayesian Dynamic Networks.
- Investigation of the level of affordable complexity for the Human- Machine Interface within the proposed Service-oriented architecture where safety critical tasks (interaction) have to coexist with tasks of a lower level of criticality (3D map representation).

This sub-task contributes to the common WP7 deliverable (D7.2), common T7.2 deliverables (D7.5 and D7.6) and an internal report for Situation awareness systems and HMI.

Relation to business needs: BN_WP7_T7.2_02, BN_WP7_T7.2_05, BN_WP7_T7.2_06
Relation to WPs: WP1. WP3, WP5 and other Use Cases of the Automotive Living Labs

ST7.2.5 Navigation and Decision-making

This sub-task is in charge of developing the decision system for the supervised automated driving when making complex manoeuvres, by taking into account surrounding vehicles and infrastructure information. From the information provided by on board localization and perception systems (ST7.2.2), the implemented decision system will be able to display warnings within a human-like driving context.. The main activities to be pursued within the sub-task are the following

- Introduce reactive Motion Planning considering uncertainties to take advantage of the Bayesian framework used all through the data flow of the subsystems developed in ST7.2.3 and ST7.2.3.
- Evaluate mixed criticalities handling by running self-Optimization and On-line Learning tasks within the same functional component in charge of the Decision System, which is intrinsically time-critical.
- Investigate emergency human-in-the-loop control strategies to properly allow the transition between human driving and automated driving
- Implement through a traffic simulator DR and IPM functionalities and evaluate its proper integration with the embedded navigation system and the HMI.

This sub-task contributes to the common WP7 deliverable (D7.2), common T7.2 deliverables (D7.5 and D7.6) and an internal report describing Navigation and Decision-making systems.

Relation to business needs: BN_WP7_T7.2_02, BN_WP7_T7.2_05, BN_WP7_T7.2_06
Relation to WPs: WP1. WP3, WP5 and other Use Cases of the Automotive Living Labs

ST7.2.6 Integration, Demonstration & Evaluation

This sub-task deals with systems integration, the setting-up and execution of the selected demonstration means: simulating the embedded system in an ADAS environment simulator where HIL is possible.
The validation and optimization of the architectural design and system behaviour in the selected scenarios will be also assessed at this stage, as previously stated in co-simulation . Special attention will be given to dependability and safety. The main activities to v b developed are:

- Integration of techniques and tools developed or customized within this use case (or in related WPS) to facilitate demonstration and evaluation of the UC.
- Iterative SW/HW integration within EMC^2 SoA architecture of the developments performed in ST7.2.2- ST7.2.5

- Technical and functional validation according to evaluation criteria, metrics and methods defined in ST7.2.1
- Provide inputs to future releases of relevant standards and specifications (in particular ISO 26262 and AUTOSAR)

This task will contribute to the common WP7 deliverable (D7.2), common T7.2 deliverable (D7.6), and one internal report regarding systems integration, use case demonstration and evaluation results.

Relation to business needs: BN_WP7_T7.2_01, BN_WP7_T7.2_03
Relation to WPs: WP5, WP6 and other Use Cases of the Automotive Living Labs

6.3 High level requirements and Business needs

The use case T7.2 addresses the following Business Needs in relation to the next-generation E/E architecture for commercial vehicles and a subset of the ECUs specific to the engine controllers:

Table 12: Business needs for use case T.7.2

No	Title	Description
BN_WP7_T7.2_01	Modeling and simulation environment for multicores	Enhancement of simulation environments to properly evaluate High Level functionalities within the adopted multicore SOA architecture, integrating performance and power modeling capabilities.
BN_WP7_T7.2_02	Harmonization of development tools	Availability of a set of harmonized tools that allow to easily integrate the design and development work flow, from system to component/code level.
BN_WP7_T7.2_03	Effective HW/SW Integration in heterogeneous platforms	Exploration of methods and tools able to increase confidence (reduce risk) of problems being observed at integration time, taking into consideration all SW layers.
BN_WP7_T7.2_04	Resource Optimization	Dynamic optimization of computational resources, hence energy consumption and costs, according to the performance requirements without putting quality at risk.
BN_WP7_T7.2_05	Effective Portability	Characterization of software components on one hardware platform so that they can be located on different hardware platforms without running the full set of timing and analysis on each platform; this supports a higher flexibility in the design of the system.
BN_WP7_T7.2_06	Parallelization and classification	Development of functionalities able to (i) recognize computing structures of application source code that may be suitable for parallel execution and (ii) to classify and evaluate different parts of the algorithms and decide if they can be effectively moved to any accelerator component of the current HW.

The High Level Requirements derived from the Business Needs are shown in the table below. (The numbering of the requirements is consistent with the numbering at project level.) While the High Level Requirements have been provided to the relevant technical WPs, these requirements will also be addressed to a large extent within the T7.2 itself.

Table 13: High-level requirements of T7.2

Requirement ID	Short description	Description	BN relation	Sub-task relation
HL-REQ-WP07-001	Modeling and simulation environment for multicores	Modelling and simulation tools should allow to compose and validate functionalities at system and code level, guaranteeing the consistency and	BN_WP7_T7.2_01	ST7.2.1, ST7.2.2, ST7.2.6

Requirement ID	Short description	Description	BN relation	Sub-task relation
		repeatability of the simulated scenarios against well-known data sets in well-documented deployment environments.		
HL-REQ-WP07-005	Harmonization of development tools	A set of seamless integrated tools for simulation should facilitate a reduction of the development, validation and deployment times for Highly Automated Driving Systems.	BN_WP7_T7.2_02	ST7.2.2, ST7.2.3, ST7.2.4, ST7.2.5
HL-REQ-WP07-006	Effective HW/SW Integration in heterogeneous platforms	The integration complexity in heterogeneous platforms (multi-core, GPGPU, FPGA) should be reduced with the help of different methods/guidelines at different phases of development	BN_WP7_T7.2_03	ST7.2.6
HL-REQ-WP07-002	Resource Optimization	The adopted SoA should provide dynamic optimization mechanisms to exploit at best the available resources in the selected HW. The resulting scheduling strategy should guarantee the fulfilment of safety critical or less critical restrictions, explicitly taking into consideration uncertainty and variability sources.	BN_WP7_T7.2_04	ST7.2.2
HL-REQ-WP07-004	Effective Portability	The modelling and development tools should allow to design SW components that are as much as possible HW independent, contributing thus to enhance the reusability and portability of the embedded functionalities.	BN_WP7_T7.2_05	ST7.2.3, ST7.2.4, ST7.2.5
HL-REQ-WP07-003	Parallelization and classification	The modelling tools should be able to accelerate the process to identify and classify parallelization opportunities given a specific component.	BN_WP7_T7.2_06	ST7.2.3, ST7.2.4, ST7.2.5

6.4 Exploitation

The development of the perception system architecture and it´s integration in heterogeneous platforms (multicore, GPGPU, FPGA) is a preliminary step to its application in Advanced Driver Assistance Systems (ADAS) design, achieving and optimal symbiosis between last generation SW components and novel autonomous qualified embedded computing platforms. ADAS are gaining more importance within today´s automotive industry. Car manufacturers and members of their value chains are constantly searching for the most advanced technologies as the requirements have changed over time and the consumer wants to drive intelligent vehicles. The platform obtained after our first implementation of the multisensory perception system on a Tegra K1 can be exploited through ADAS Tier 1 to design high performance automotive embedded systems. The natural target user of our system, the 3CCAR warning-based enhanced situation awareness system for urban environments, are Tier 1 ADAS suppliers.

HIB exploits the outcomes of this Use Case – Highly Automated Driving to evolve its solutions around the traffic simulation framework (SUMO), on which it has been actively working in the last years. Thereby, this traffic simulator now allows bidirectional connectivity between the Traffic Simulation Platform and ADAS (Advanced Driver Assistance Systems) simulator with frequent updates. Thereby, the highly automated vehicle and its surrounding relevant elements will be generated by the platform, whose local dynamic behavior will be updated according to the ADAS simulator feedback.

7. Appendix 3 T7.3 UC Design and validation of next generation hybrid powertrain / E-Drive

7.1 KPIs

Table 14: KPIs for use case "Design and validation of next generation hybrid powertrain / E-Drive"

No	Title	Description	Sub-task relation
KPI_T7.3_01	Increased verification coverage by simulation	Increase verification coverage that can be performed in simulation by 10% for highly parallel applications and thus reduce the validation effort on target HW	ST7.3.1, T7.3.5
KPI_T7.3_02	Benefits of a seamless modeling method	Improve design specification maturity and consistency over the different domains, thus reducing cost and time for system design and safety recertification by 15%	ST7.3.2
KPI_T7.3_03	A collection of SW design and development guidelines for multicore platform	Prepare for each SW layer minimum 1 document including guidelines for the design and development of this component on multicore platforms, thus reducing training time for new SW engineers	ST7.3.1, T7.3.3
KPI_T7.3_04	Tailored safety architecture	Defined safety mechanisms for multicore to cover all automotive safety levels	ST7.3.1, ST7.3.2, ST7.3.3
KPI_T7.3_05	Decrease development time by use of integrated toolchain	Reduce development time at tool boundaries by 10% by efficient data transfer and consistency check	ST7.3.1, ST7.3.4, ST7.3.5
KPI_T7.3_06	Safety case generation	Reduction of effort for safety case generation by 20 %	ST7.3.3, ST7.3.4
KPI_T7.3_07	Increased data throughput	Increase data throughputs of existing communication channels by the development and integration of new technologies such as CAN FD, Ethernet	ST7.3.6
KPI_T7.3_08	Data exchange and communication strategies	Maintain same configuration and handling efforts for communication complexity / load increase of 50%	ST7.3.6

7.1.1 **ST7.3.1 UC_Electrical and functional integration of high dynamic e-drive system controls with time based vehicle control algorithm**

7.1.1.1 Short description

Multi-core processor systems offer a substantial potential for integrating diverse calculation tasks /calculation algorithms on one computing unit which are currently executed on two different controller units. This is especially appealing for strongly cost-driven automotive industry as it enables the economization of one embedded controller unit.

The development target of this task therefore is a cost-efficient, robust and reliable runtime environment for electric powertrains in HEV-, PHEV and EV-applications which combines the functionality of an E-Motor Controller Unit (EMCU) with the Vehicle Controller Unit (VCU) in one automotive controller unit (VEMCU). By investigating the capabilities of the multi-core technology in one concrete application a bottom-up approach is followed, as the results acquired will be integrated in generally valid functional design guidelines. Goals of this task are:

- to provide requirements and specifications for the application
- to perform / support integration of the different tailored technologies (outcomes from the other tasks)
- to perform evaluation of the proposed EMC² technologies

Relation to business needs: BN_T7.3_01, BN_T7.3_03, BN_T7.3_04, BN_T7.3_05
Relation to WPs: WP1, WP3, WP4

7.1.1.2 Spotlight outcomes

Ontology-based Runtime Reconfiguration

An application can be extended by a service we refer to as ontology-based runtime reconfiguration (ORR) which substitutes failed services. ORR increases the availability of the system by exploiting implicit redundancy (see details in D7.7 and D7.8). While the mechanism has been developed in WP3, we investigated the requirements regarding the underlying architecture, the applicability and setup of the mechanism on the example of this task's application. Finally, we evaluated the platform running ORR and give potential improvements.

Activities and results achieved in this subtask:

- Architectural requirements to enable ontology-based runtime reconfiguration (ORR) were defined. ORR adapts a system's structure (adding, removing or reconnecting components) to substitute failures by implicit redundancy.
- The evaluation of ORR shows that a suitable underlying architecture simplifies integration and increases the applicability of ORR. For instance, in a system running a SOA framework (service management) and communicating via publish/subscribe, structural adaptation like ORR can be completely decoupled from the application, i.e., existing application components can be reused and don't have to be altered to enable structural adaptation.
- The runtime evaluation shows that the execution time of ORR is negligible compared to the overhead produced by the service management (starting, stoping and connecting services) of our prototype. Once the substitute for a failed service is running, the performance is only limited by the computational ressources of the platform.

7.1.2 ST7.3.2 UC_Model-driven system / software / hardware / safety engineering for embedded multi core hybrid powertrain and e-Drive control systems

7.1.2.1 Short description

Goal of this task is to enhance model-driven systems engineering approaches in order (1) to better take into account the multi core aspects during system / software / hardware / safety engineering, (2) propose seamless (architecture) modeling approaches to minimize the specification gap between the different disciplines, and (3) improve the links between specification and development tasks. This work covers the following aspects:

- Enhancement, tailoring and integration of semi-formal system / architecture description languages (e.g., SysML, CESAR Meta Model, EAST-ADL, AUTOSAR) in order to provide a unified framework for the seamless specification of embedded systems

- Efficient integration of integrated safety mechanisms into an overall concept. This includes the seamless development of the safety concept as well as its verification (dedicated analysis) and validation (test).
- Developing an automotive domain ontology to reveal the semantics of the interaction between different controller units. This ontology will enable integrating mechanisms to compensate unit failures.

Relation to business needs: BN_T7.3_02, BN_T7.3_04
Relation to WPs: WP1, WP3

7.1.2.2 Spotlight outcomes

The application, here, an electric vehicle control, integrated from various T7.3 partners has been modeled using the service-oriented architecture (SoA). Usually, a system can be divided into several components, each implementing various services. ORR can be (additionally) connected to the system which here communicates via CAN, though other communication principles are possible (see Figure 30).

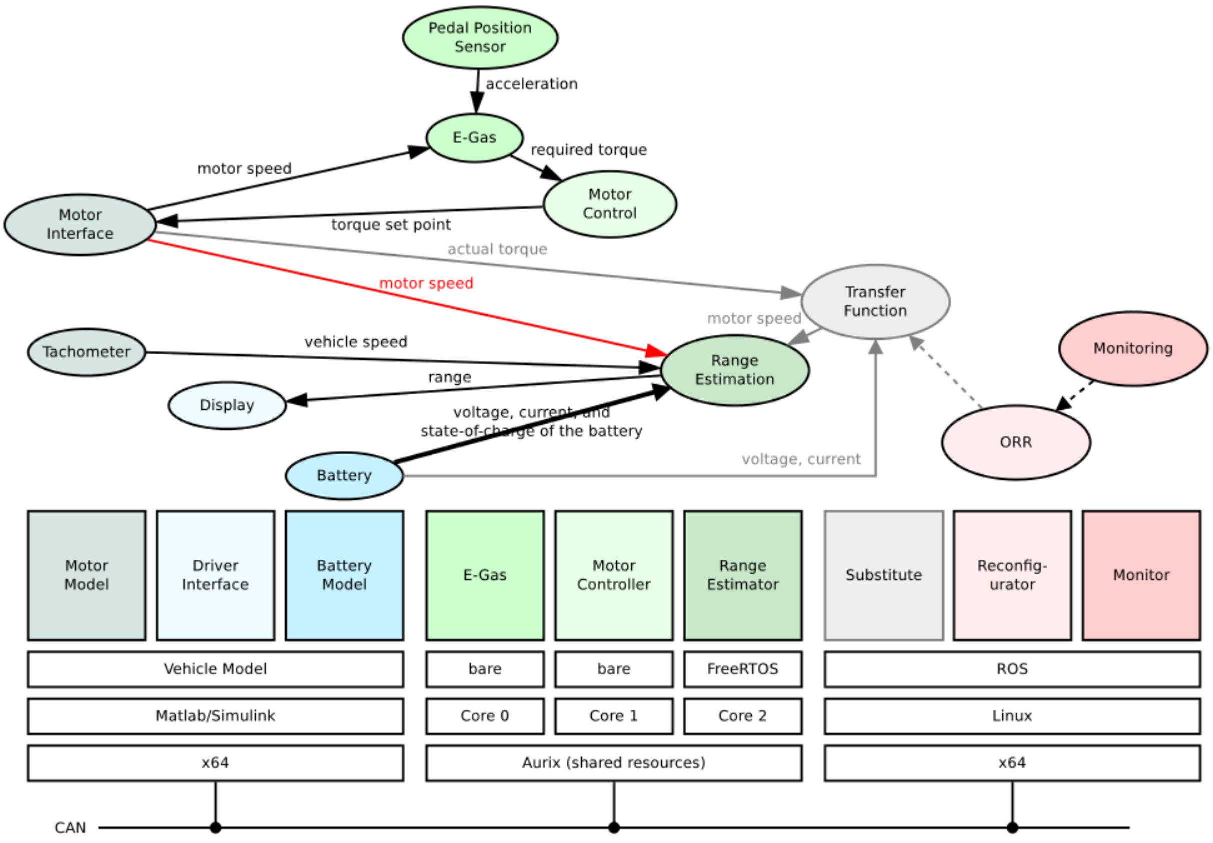


Figure 30: Component- and service-oriented view of the demonstrator. Subsystems and components are depicted as rectangles (stacked on each other to visualize hierarchy), whereas services are represented by ellipses. Exchanged information is represented by arrows.

For instance, a multi-core device "Aurix" (a subsystem) includes two cores running the e-motor controller and a component estimating the remaining range of the vehicle. The electric vehicle is simulated in Matlab controlled by the output of the multi-core device and providing measurements back. The services exchange information about the system. Typically, the exchanged information are valued properties of the system like state variables or measurements (e.g., vehicle speed). The relationships between properties are modeled in a knowledge base we refer to as ontology (see Figure 31). Using the ontology failed components can be substituted, e.g., a failed motor speed sensor.

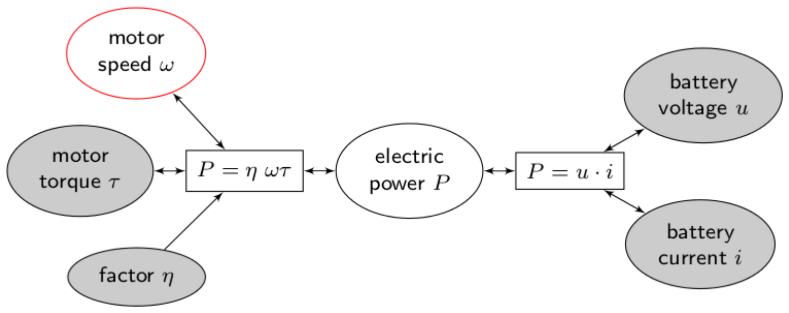


Figure 31: Exemplary ontology of the use case (only parts related to the application and reconfiguration scenario are depicted). The shaded ellipses are properties provided by services (e.g., sensors). In our scenario the service providing the motor speed fails, denoted as a red ellipse.

PIL simulation was implemented between Matlab Simulink and selected AURIX TC277 processor. PIL simulation is important part of model based design and it also belongs to rapid prototyping tools. It enables to validate how the developed code will perform in a real target hardware. In this case, it controls the PMS motor model which is running in Simulink. The communication is realized using SCI interface which runs through the same USB cable which is used for program loading and debugging. PIL helps to determine sufficiency of data representation in target hardware. Moreover, it enables to test the control algorithm in all working points and to provide code profiling in these situations. Thus, it is possible to check computation burden in each sampling period, not only maximum, average or some instantaneous value. Implemented PIL simulation is interrupt driven so it is non-blocking for the core where it is executed. It enables to test execution times while sharing resources between several cores. In AURIX these are peripherals and memory.

Next to this, a SysML system approach was followed to define the specification model (consisting of use-case definitions, textual requirements, sequence diagrams and block definition diagrams). Based on this specification model, a Simulink model was created. This Simulink model was thereafter imported into the Enterprise Architect (model-based) system engineering tool by using the C++ code generation functionality of Matlab/Simuling and the reverse engine eering functionality of Enterprise Architect. The imported model elements constitute the execution model. This model was thereafter enriched with “trace” and “satisfy” relationships towards the sepcifications and also with “verify” relations towards the test cases.

Activities and results achieved in this subtask:

- Development of an exemplary ontology in the automotive domain and the methodology to create the domain knowledge.
- Implementation and integration of ORR into an existing application of different partners has been shown in the WP7.3 demonstrator.

7.1.3 **ST7.3.3 UC_Programming paradigms and SW architecture for embedded multi core hybrid powertrain and e-Drive control systems**

7.1.3.1 Short description

Following topics shall be addressed:

- Programming paradigms and porting of SW applications to multi-core platforms; Impact regarding programming models, SW architectures, operating systems, task allocation, timing issues and WCET
- Efficient integration of functions and safety into an overall concept. More especially, following aspects shall be focused on:

- Concept for SW Architecture and Partitioning (re-grouping e.g. VCU and MCU on one silicon), hence also ensure coexistence of mixed-criticality functions and consider also interprocessor communication.
- Definition and Implementation of scalable safety mechanisms to target applications from ASIL A – ASIL D
- Definition and Implementation of efficient and safe library for mathematical operations and memory access

Relation to business needs: BN_T7.3_03, BN_T7.3_04, BN_T7.3_06
Relation to WPs: WP2

7.1.3.2 Spotlight outcomes

The architecture of the prototype was designed to be highly decoupled from the underlying hardware. One of the measures taken to achieve this goal was to use a real-time embedded OS. It can be seen as virtualization of the communication hardware (e.g. CAN link) from the perspective of the higher-level applications. Moreover, the prototype was developed with respect to object oriented software concepts, thus member access, inheritance and scaleability was considered as well.

With the help of the well-chosen architecture and design, the prototype is not solely dedicated to a predefined bus or hardware. On the contrary, the implementation can be extended to function with multiple networks and hardware without major code rewrite. Furthermore, the inter-core communication (self-made extension of FreeRTOS) was implemented in a way so that there is no limitation to the number of involved cores.

7.1.4 **ST7.3.4 UC_ Safety case support for the development of embedded multi core systems in mixed criticality context**

7.1.4.1 Short description

Goal of this task is the preparation and documentation of the argumentation why the system developed is reasonably safe / reliable for the purpose it is intended to. This task is related to the ISO 26262 Part 8 Clause 11 (Confidence in the use of software tools) and ISO 26262 Part 4 Clause 10 (Functional safety assessment – safety case). The edrive system will be study as a SeooC (Safety Element out of Context) and the implications of treating it like that.

This task covers two aspects:

- Confidence in use of single tools as well as integrated tool chains; how far the tools / tool chains using during the development are reliable enough in order to support the development of safety-critical embedded systems and the needs for qualification.
- Safety case support: Automated extraction of development information and traces (e.g., safety analysis, safety requirements, test cases) to support the correctness, completeness and consistency of the safety aspects of the development. Develop safety case patterns in relation with the multi core development to support the argumentation. Apply modular safety cases strategy to the safety case arquitecture.

Relation to business needs: BN_T7.3_05, BN_T7.3_06
Relation to WPs: WP6

7.1.4.2 Spotlight outcomes

The amount of ECU’s within vehicles is increasing. Upcoming multi core technology can possibly stop this trend by combining different control applications into one multi core ECU. Ideally this multi core ECU’s are usable for mixed critically applications.

With help of the presented AVL use case "Design and validation of next generation hybrid powertrain / E-Drive", the design and validation of several system architectures based on multi core platforms should be investigated. The use case provides two specific use case scenarios which deal with "Safety Contracts" (contributors: AVL, IESE, TEC) and "Safety and Security Cases" (contributors: AVL, VIF, AIT and TEC) (see Figure 32).

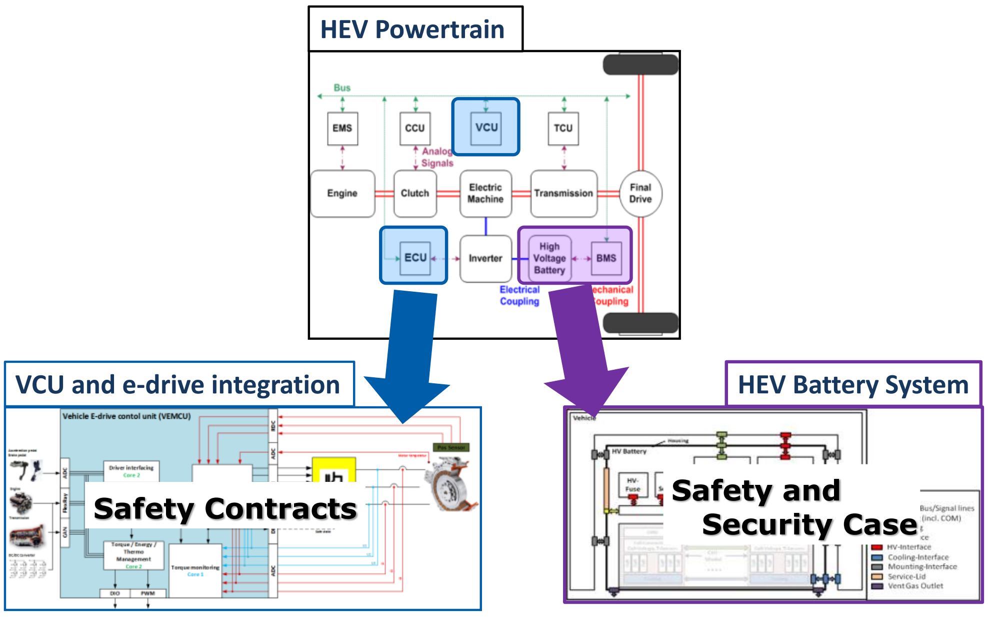
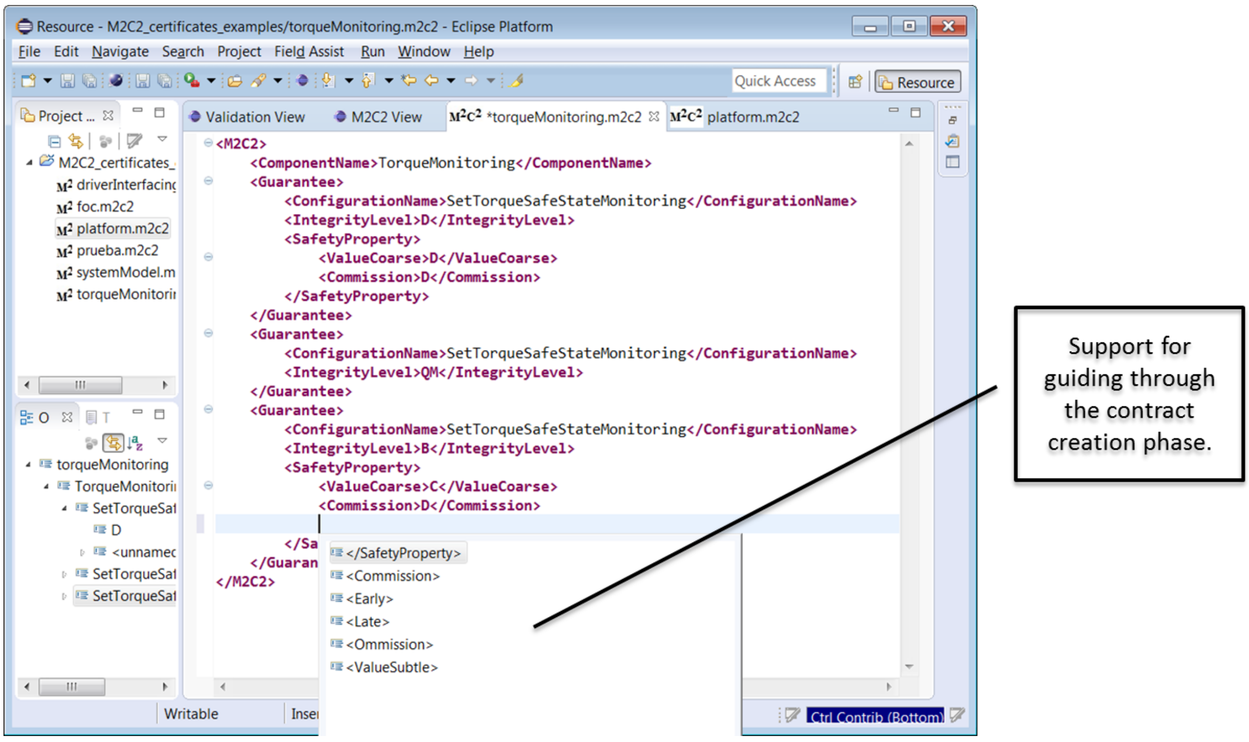


Figure 32: Overview of scenarios of AVL use case

7.1.4.2.1 Scenario 1 - Safety Contracts

The multicore capabilities of the "Vehicle E-drive control unit" (VEMCU) allows multiple applications to be deployed in a single piece of hardware. Systems as such are envisioned to be developed by different manufacturers and, due to updates, their final configuration (i.e. combination of applications and platform) can be surely known only at runtime. To address this phenomenon, we propose ConSerts M (former M2C2), a contract based runtime assurance approach designed at development time and evaluated at runtime by the systems themselves (i.e. without the need of further human intervention).

Activities performed within this subtask relates to scenario 1 as follow:
- Specification of the Multidimensional Modular Conditional Certificates (M2C2) (renamed to ConSert M) defined in WP6 for the use case environment
- Use of the tool created for M2C2 contracts edition (see Figure 33).
- Apply the M2C2 API to real time verification of the safety and dependability constraints



```
<M2C2>
    <ComponentName>TorqueMonitoring</ComponentName>
    <Guarantee>
        <ConfigurationName>SetTorqueSafeStateMonitoring</ConfigurationName>
        <IntegrityLevel>D</IntegrityLevel>
        <SafetyProperty>
            <ValueCoarse>D</ValueCoarse>
            <Commission>D</Commission>
        </SafetyProperty>
    </Guarantee>
    <Guarantee>
        <ConfigurationName>SetTorqueSafeStateMonitoring</ConfigurationName>
        <IntegrityLevel>QM</IntegrityLevel>
    </Guarantee>
    <Guarantee>
        <ConfigurationName>SetTorqueSafeStateMonitoring</ConfigurationName>
        <IntegrityLevel>B</IntegrityLevel>
        <SafetyProperty>
            <ValueCoarse>C</ValueCoarse>
            <Commission>D</Commission>

        </Sa
    </Guaran
</M2C2>
```

Figure 33: Screenshot of the tool while creating the M2C2 contract

The contributions related to the use case scenario are divided in two demonstrators:

- 2nd year demonstrator: the use case was applied to demonstrate the eclipse-based tool (cf. Figure 33) created with the purpose to help the safety engineer to build syntactically-correct contracts. With the tool, the engineer can also verify the contracts compatibility (i.e. whether the guarantees and demands fit each other). Using the tool, contracts were created for the "VCU and e-drive integration" architecture as proposed at the beginning of the project. The use case scenario allowed the researchers to implement the ideas to a real application.

- 3rd year demonstrator: a Runtime Mediator (i.e. a piece of software designed to verify the ConSerts M contracts at runtime) was developed and integrated with the implemented architecture. The idea was to check, through contracts, if the software composition can deliver its services with enough safety reliability before being deployed onto the target platform. The development of the Runtime Mediator and the enhancement of the capabilities of the contract-language were outcomes of this work.

Additionally, the use case scenario was presented as case study in the book chapter "*Runtime Safety Assurance for Adaptive Cyber-Physical Systems: ConSerts M and Ontology-based Runtime Reconfiguration Applied to an Automotive Case Study*" from the "*Handbook of Research on Solutions for Cyber-Physical Systems Ubiquity*". The scenario illustrated the application of ConSerts M and Ontology-based Runtime Reconfiguration.

The aforementioned activities are aligned with KPI_T7.3_05 *Trustworthiness of ACC system architecture* and KPI_T7.3_06 *Degree of contribution to the development of sustainable mobility and transportation*, thus, contributing to further development of technology and knowledge.

Evaluation scenario 1

The Table 15 summarizes the evaluation of the respective KPI towards the scenario 1.

Table 15: ST7.3.4 KPI evaluation of scenario 1

No	Title	Description	Assessment of KPI fulfilment
KPI_T7.3_05	Decrease development time by use of integrated toolchain	Reduce development time at tool boundaries by 10% by efficient data transfer and consistency check	ST7.3.4 Scenario1: The tool eases the creation of contracts and allows syntax checks, enhancing the efficiency of the safety engineer. The KPI is largely fulfilled.
KPI_T7.3_06	Safety case generation	Reduction of effort for safety case generation by 20 %	ST7.3.4 Scenario 1: Using contracts allow reuse of safety engineering efforts and its automated evaluation allows compatibility of components not considered at development time. The KPI is largely fulfilled.

7.1.4.2.2 Scenario 2 - Safety and Security Case

The large amount of functions and ECU's in today's cars need a lot of communication. Information has to be exchanged between the ECU's and between ECU and environment, but communication potentially contains threats. To prevent exploitation of such threats cybersecurity processes have to be applied.
The execution of software related to different functions on one ECU will probably cause influence between these functions. If only one of these functions has to cover safety aspects the potential of possible influence has to be known in detail. Furthermore the developer has to make sure that safety measures will be effective at any time. To guarantee safety integrity for a product ready for the market, developers already have to deal with safety aspects and possible influence in early development phase e.g. in the concept phase. Most domains require a safety case as a proof that safety was considered during the development and a sufficient risk reduction was reached. The generation and documentation of safety cases for complex systems is already a very time-consuming task. Adding security increases the required effort and complexity. Security adds not only a second property to consider and to argue, but also interactions and trade-offs. A safety and security case needs therefore to argue why a system is safe, even when facing malicious intentional actions and that security has no negative impact on safety.
The second use case scenario "Safety and security case" deals with the question how to integrate safety and security aspects in the development process.

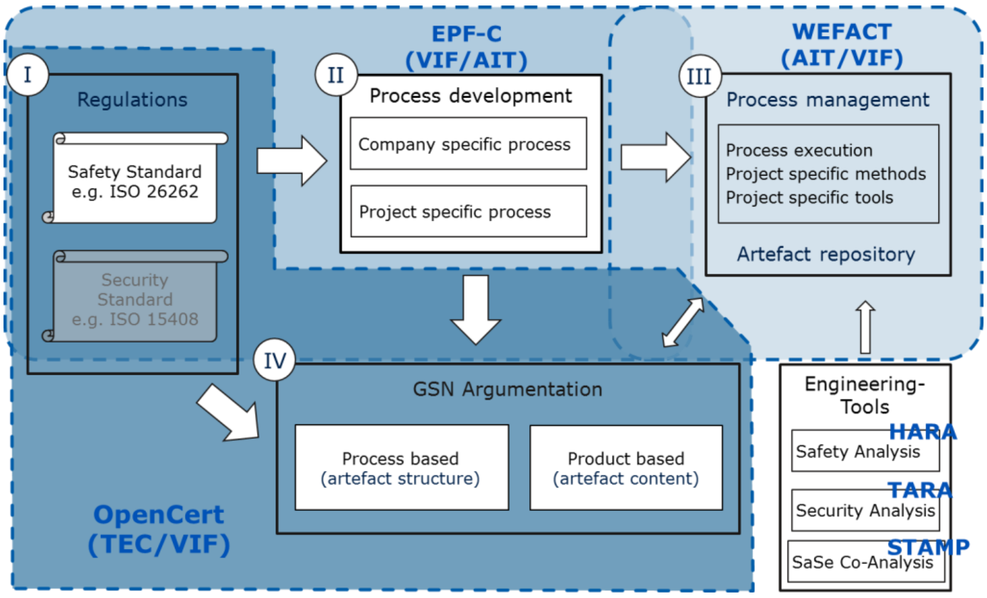


Figure 34: Overview of the safety and security case methodology (tool coverage in blue)

Figure 34 shows an overview about the methodology related to the scenario 2.
Activities performed within this subtask relates to scenario 2 as follows:

- Model the ISO 26262 standard (EPF-C)
- Process management (WEFACT)
- Safety and security co-analysis (STAMP, FMVEA)
- Interoperate different tools (OpenCert, VISSURE)
- Applying the OpenCert tool and define an assurance project for the use case.

Modelling of the ISO 26262 standard (EPF-C)

Regulations are often provided as standards, which provide the most relevant requirements. For example ISO 26262 provides mandatory requirements concerning safety and SAE J3061 provides cybersecurity process framework and guidance. Additionally they contain associated work products. For the daily work within a development project it will be favourable to use a model of the standard (see Figure 35). The advantage is that all requirements are represented as tasks connected with their input and output work products. The standards have been modeled based on an approach created in Opencoss and SafeCer project.

The EPF-Composer is used to create a process model related to a standard and it provides additional features as well. Process developers can add company specific aspects and tailor generic processes to a project specific shape. The produced project specific process gains profit from the existing experienced company specific process because a large number of guidance and templates can be reused without or with minor changes. The EPF-Composer compiles a linked model, which is easy to use even for users with little experience with standards the model is based on.

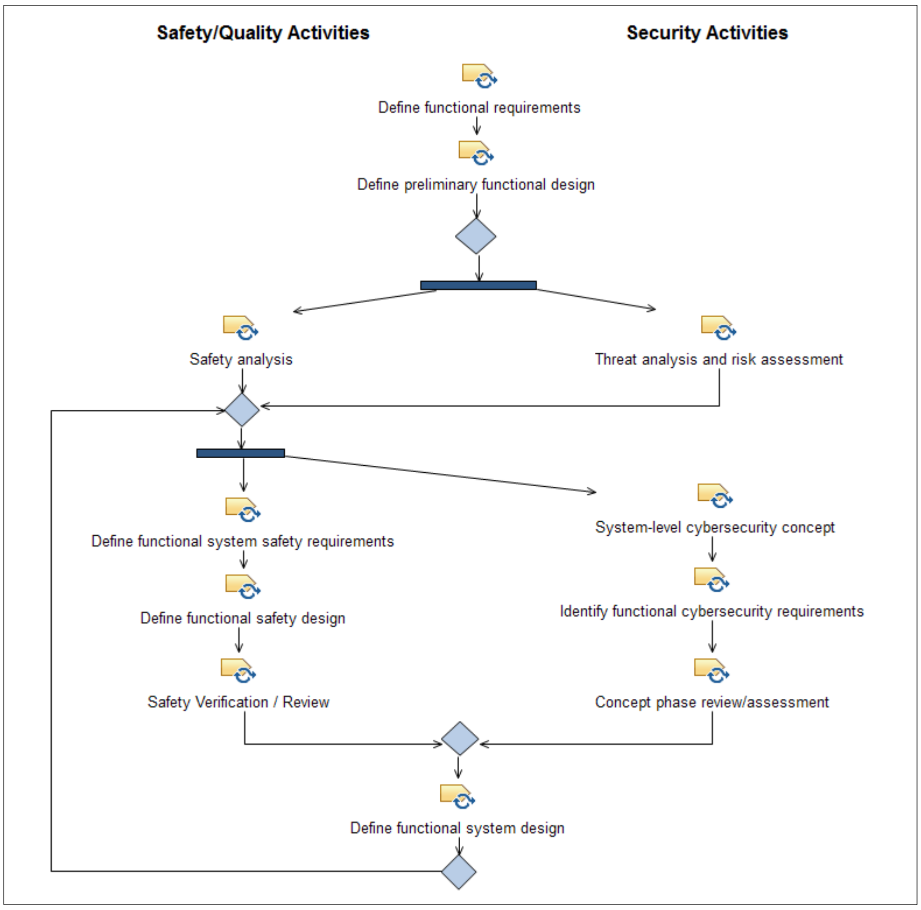

Safety/Quality Activities
Security Activities
Define functional requirements
Define preliminary functional design
Safety analysis
Threat analysis and risk assessment
Define functional system safety requirements
System-level cybersecurity concept
Define functional safety design
Identify functional cybersecurity requirements
Safety Verification / Review
Concept phase review/assessment
Define functional system design

Figure 35: Model of concept phase concerning safety and security activities

Process management (WEFACT)

WEFACT is an automated tool which supports the generation of safety case by automating the execution of V&V Tools and managing the evidence. Final result of the WEFACT supported workflow is the safety case. WEFACT's requirements tracking, testing and certification support is based on a workflow derived from the requirements of functional safety and cybersecurity standards, but also other domain specific requirements and company-specific practices can be included. These requirements, together with functional and non-functional requirements defined for the individual application, are stored in a database; so-called V-plans (validation plans) are defined for all these requirements and their successful execution finally proves that the requirements are fulfilled.

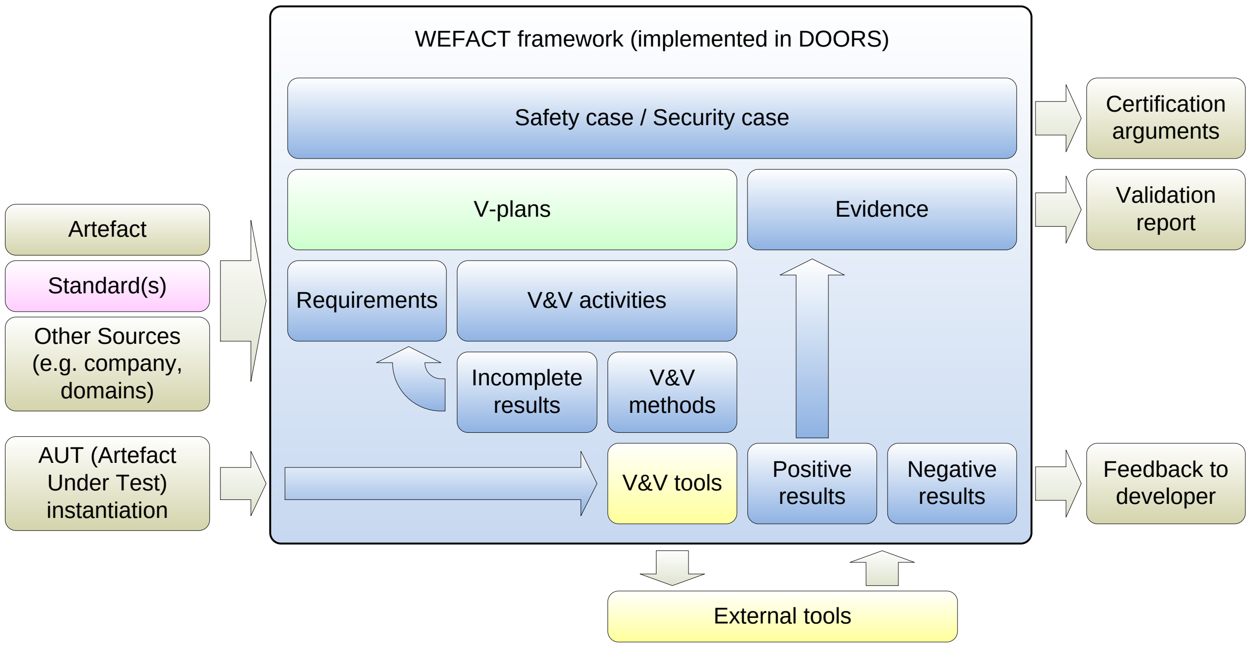


Figure 36: Depiction of WEFACT and its Interfaces

The V-plan for the AUT describes the responsibilities and activities, based on safety and security standards as well as other sources like domain-specific and company-specific practices. WEFACT guides through the combined process of achieving certification according to safety and security standards. In addition, activities for verification and validation (V&V) are connected to external tools which can be integrated into the workflow engine. WEFACT supports automated tool integration over Open Services for Lifecycle Collaboration (OSLC), an interoperability standard for the cooperation of lifecycle management tools. Depending on the level of integration, i.e. whether the V&V tool can be called directly via OSLC or command line interface, or a V&V activity needs manual interaction with external tools, WEFACT will be able to conduct the V&V activity more or less automatically and change the requirement status according to the result (<pass> or <fail>). After all V&V activities of the V-plans are conducted successfully and all requirements are therefore fulfilled, a holistic safety and security case is generated. This so called dependability or assurance case uses an argument notation like for instance the Goal Structuring Notation (GSN) to demonstrate the assurance that a system is safe and secure.

Safety and security co-analysis

We have investigated the "Cybersecurity Guidebook for Cyber-Physical Vehicle Systems" SAE J3061 and created a template for security classification based on the EVITA methodology.
Additionally the process- and product- based argumentation was extended to cybersecurity.
Established safety analysis methods don't consider security threats explicitly as a cause for safety hazards. There are currently two approaches to solve this. One is to extend established safety analysis

techniques (FMVEA, BDMP, CFT, and SAHARA) with new features. The other is to develop new analysis techniques (CHASSIS, STPA-SEC)

In EMC^2 we applied STPA-SEC, FMVEA and SAHARA to the Battery Management System (BMS):

- Systems-Theoretic Process Analysis-Security (STPA-SEC) is based on a System-Theoretic Model of Accidents (STAMP) by Nancy Leveson. STAMP is a new approach to understanding accidents, based on system theory. Systems are understood as interacting elements with multiple feedback loops, exchanging information and control commands. Accidents are caused when safety-related controls and constraints were not in place or did not prevent or detect maladaptive changes. In other words, the absence of controls and constraints in a system can lead to an accident. The STPA-SEC approach is an interesting addition to the existing safety and security engineering toolbox. In our experience it is especially suited for software dependent systems, where deviations between the perception of the process and the process itself could cause critical consequences. But since the support for the analysis of other causes (security, component failures and external influences) is rather sparse STPA-SEC should not be used as the only method.
- FMVEA extends the FMEA method with threat and vulnerability modeling and allows a combined elicitation of failures and threats. A system model is combined with a threat and failure database in order to identify potential threats and failures for each element of the system. Potential effects of failure and threats are identified and based on the severity of the impact and the likelihood the risk is calculated. Similar to FMEA it is difficult to identify hazards which are caused by a combination of failures or threats.
- SAHARA extends the ISO 26262 HARA method with STRIDE and allows the identification of hazards and threats based on a prelaminar system design. The risk assessment is compatible with ISO 26262 and there is a predefined communication path from security to safety. Based on the identified risk (high likelihood or safety impact) security threats have to be communicated to the safety team and receive an ASIL assignment.

Interoperate different tools

Automatic generation of compliance requirements related to ISO 26262 goals for the use case into a requirements management tool. Interoperate different tools, a compliance management tool (OpenCert) and a requirements management tool (VISURE-Requirements) applying OSLC technology defined in WP5.

Applying the OpenCert tool

Elaboration of process- and product- based safety and security argumentation. Use of OpenCert tool to create GSN structures which are related to work products and requirements as a part of the assurance case. Based on this methodology we can create fitting GSN structures (see Figure 37) for both types of argumentation, which deal with functional safety and cybersecurity. Semi-automatic generation of process-based arguments related to the ISO 26262 goals compliant to the assurance case has been shown.

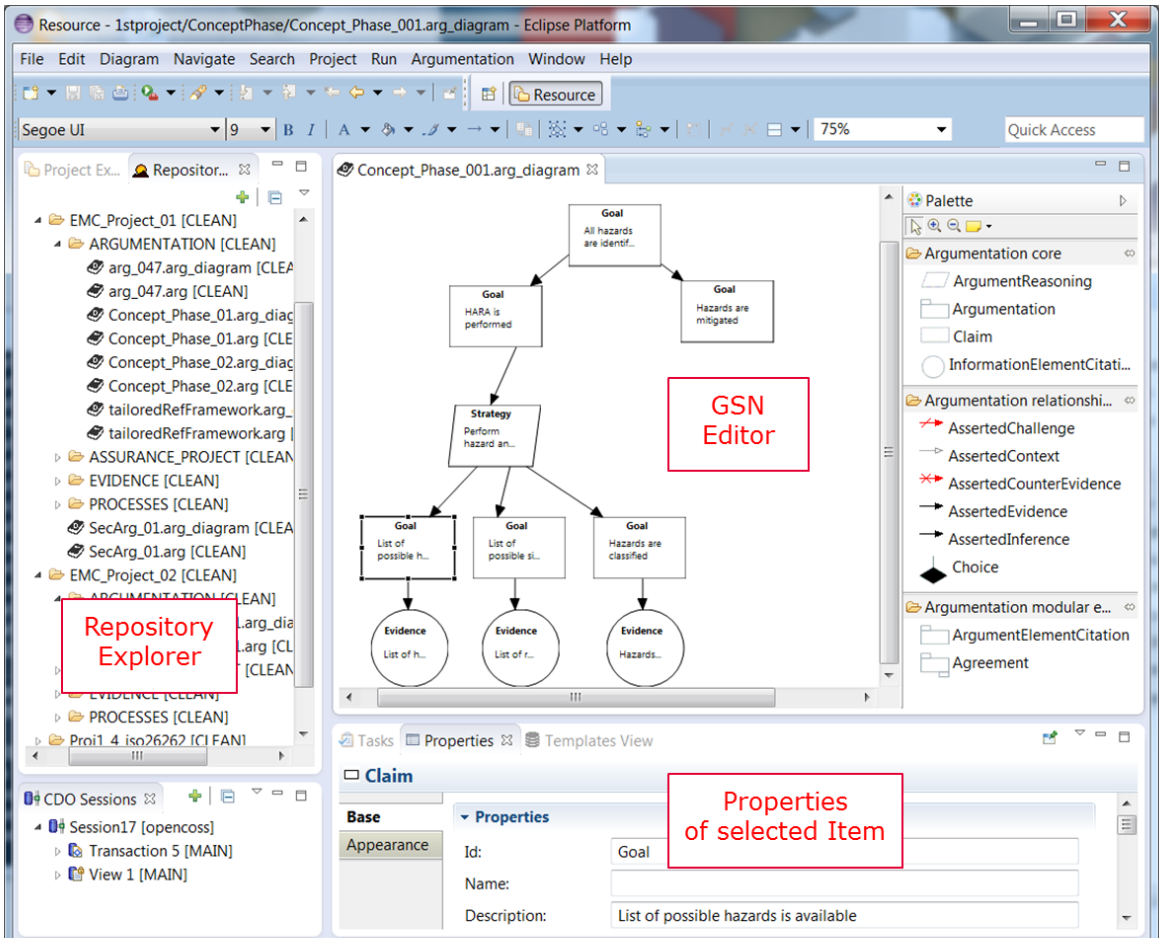

Resource - 1stproject/ConceptPhase/Concept_Phase_001.arg_diagram - Eclipse Platform
File Edit Diagram Navigate Search Project Run Argumentation Window Help
Repository Explorer
GSN Editor
Properties of selected Item
Palette
Argumentation core
ArgumentReasoning
Argumentation
Claim
InformationElementCitati...
Argumentation relationshi...
AssertedChallenge
AssertedContext
AssertedCounterEvidence
AssertedEvidence
AssertedInference
Choice
Argumentation modular e...
ArgumentElementCitation
Agreement
Claim
Base
Appearance
Properties
Id: Goal
Name:
Description: List of possible hazards is available

Figure 37: Screenshot OpenCert GUI

Evaluation scenario 2

Evaluation of tool support by applying the methodology to the use case.
Creating a model of underlying standards with links between requirements and work products makes it easy to check completeness and availability. Additionally the visualization of linked elements reduces effort as well as the opportunity to generate evidence automatically.

The use of tools (OpenCert, WEFACT, EPF-C) reduces development time what leads to the rating that the assigned KPI's (KPI_T7.3_05, KPI_T7.3_06) are fulfilled largely.

Table 16: ST7.3.4 KPI evaluation of scenario 2

No	Title	Description	Assessment of KPI fulfilment
KPI_T7.3_05	Decrease development time by use of integrated toolchain	Reduce development time at tool boundaries by 10% by efficient data transfer and consistency check	ST7.3.4 Scenario2: Model the ISO 26262 standard by EPF-C Process management by WEFACT Interoperate different tools by OpenCert and VISSURE The KPI is largely fulfilled.
KPI_T7.3_06	Safety case generation	Reduction of effort for safety case generation by 20 %	ST7.3.4 Scenario 2: Applying the OpenCert tool and define an assurance project for the use case. Safety and security co-analysis (STAMP, EVITA) The KPI is largely fulfilled.

7.1.5 **ST7.3.5 UC_ Tools and tool platforms for the design and validation of next generation hybrid powertrain / E-Drive control systems**

7.1.5.1 Short description

Goal of this task is to enhance the tooling platform and development environment in order to enable more efficient development of embedded multi core systems in a mixed criticality context. Enhancement, in this case, comprises (a) the enhancement of single development tools for multi-core control units (e.g., compiler or vehicle simulation tool taking into account high degree of parallelism of new generation control units based on multi cores), and (b) improving interoperability and integration aspects between the different development tools (providing tool chains for embedded multi core systems in mixed criticality context).

Following activities are planned:

- Provide simulation environment to support safety analysis in the context of ISO 26262 (e.g., hazard analysis and risk assessment or (E/E) system FMEA) and understand vehicle reaction in case of malfunction from elements or control systems. This shall provide higher confidence for the understanding of the impact and during the classification of the hazards.
- Co-simulation is a demand with increasing importance both in pure office applications as well as in HiL applications. Robust coupling strategies have to be applied in order to guarantee stable and performant calculation results. Besides others, these strategies involve extrapolation/interpolation, filtering, accuracy monitoring activities. Furthermore, adaptive and multi-level data exchange strategies need to be applied. For the purpose of this task, the tools CruiseM 84 and ICOS 85 will play a central role.
- Development of integrated tool chain supporting seamless model-based systems / SW / safety engineering based on a semi-formal system description (e.g., EAST-ADL, SysML). Industrial tools such as e.g., Enterprise Architect, MKS, Matlab Simulink shall be integrated

Relation to business needs: BN_T7.3_01, BN_T7.3_05
Relation to WPs: WP5

7.1.5.2 Spotlight outcomes

The AVL in house software framework “Arte Core framework” was used in order to extend functionality of the underlying real-time operating system INtime. Moreover the third-party open source real-time framework “Boost” was used in order to build a ring buffer.
For the validation of the prototype the CAN bus measurement tool PCANView from PEAK-System was taken, in order to observe the CAN bus data traffic. In addition, the very similar tool CANalyser was helpful to proof the proper functioning of the prototype.

The integration and validation of more than one software stack on a multi-core device requires new methodologies with respect to core to core communication, core synchronisation or access to shared resources and makes special demands to the used toolchain. To cover this challenge a toolchain with ICOS as central part was provided.

ICOS, an Independent CO-Simulation platform covers a wide range of the evaluation/simulation process.
Form the first simulation of a software e.g. developed in Matlab/Simulink (MIL) as single software stack or together with other software and plant models until the first test on the hardware as single stack on one core or with different stacks on several cores.

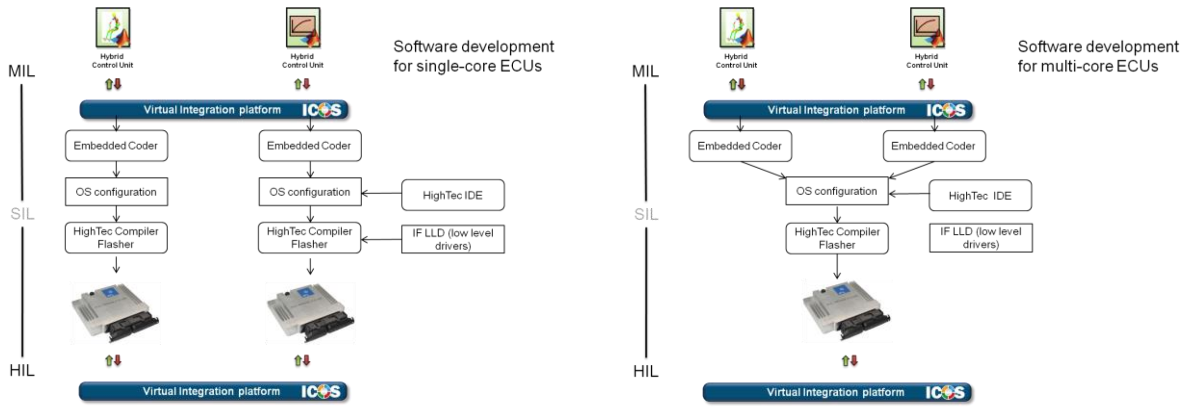

Software development for single-core ECUs
Software development for multi-core ECUs
MIL
SIL
HIL
Hybrid Control Unit
Virtual Integration platform
ICOS
Embedded Coder
OS configuration
HighTec Compiler Flasher
HighTec IDE
IF LLD (low level drivers)

Figure 38: SW Development Workflow with ICOS

The left figure above shows a typical scenario for single-core software development simulated with ICOS. The right picture above shows the implemented approach in this use case. Software developed from different software engineers is fused, configured, built and evaluated in a seamless toolchain.

Tools integrated in the toolchain:

- Matlab/Simulink: modeling and software development
- Embedded Coder: c-code generation
- Hightec IDE: c-code development, os implementation, os configuration, middle implementation
- GCC compiler: c compiler
- PLS Debugger: debugging, tracing
- ICOS Independent CO-Simulation Platform: Co-simulation models, software, hardware with plant models
- PICan: CAN communication ICOS <-> Matlab/Simulink <-> AURIX TC277 <-> ROS nodes

Because of necessity for a better evaluation of the toolchain, the AVL use case "Design and validation of next generation hybrid powertrain / E-Drive" was expanded. Additional software from BUT, TUW, VIF, and TNO has been developed (with tools from the toolchain). (E.g. TNO provides battery management software and predictive range estimation algorithm where the driving behaviour of the driver is the input. BUT provides a control strategy for an additional e-motor and AVL an e-gas software stack.)

The additional software was implemented in a distributed development environment (differned vendors using certain tool(s) form the toolchain for their development), simulated, tested and integrated on an AURIX TC277 based on the toolchain described above.

The safety security toolchain described in ST7.3.4 expands the programming toolchain on the top.

Evaluation done in the use case:

- Evaluation of the toolchain: by software development and integration for a multi-core target device (AURIX TC277): Matlab/Simulink <-> Embedded Coder <-> Hightec IDE <-> GCC Compiler <-> Flashtool <-> Tracetool/Debugger <-> ICOS
- Investigation of the behavior of the application(s) software running on a automotive multi-core device (AURIX TC277) against their requirements with respect to timing and safety. - ICOS, Tracer, Debugger
- Investigation of the internal behavior of the sytem concerning shared resources with resect to timing and safety (shared mem, basic software running on one core - accessed by all cores (COM Stack), core synchronisation) – ICOS, Tracer, Debugger
  Investigation of the internal behavior of the middleware with respect to timing and safety (freeRTOS implementation, running on all cores) - ICOS, Tracer, Debugger

An integrated tool chain supporting seamless model-based testing was also pursued in an effort to increase test coverage while at the same time not increasing the test efforts (and costs). For this purpose a recommended test approach was defined, consisting of the following guidelines:

- Separate the system under test from its test environment (separation of concerns). Make the communication interfaces between the system under test and the test environment explicit;
- If it is expected that the test-harness needs to be ported onto dedicated hardware, then the test-environment shall be separated into a test-suite and a test harness. Also in this case it is preferred to already include the communication blocks that will be required for communication once the functionality is mapped onto the envisioned (test) hardware;
- Select the test platform that is most appropriate (specifications) for verifying the requirements for the system under test. Notice that often a non real-time system can be used for verifying real-time behaviour. A few restrictions need to be considered however:
    1. The test platform to be used for verifying the real-time requirements for the system under test shall provide a low (enough) latency time stamp mechanism for labelling events received from and sent to the system under test;
    2. If events can be missed due to non-real-time behaviour of the test platform, then the test platform shall be able to detect and report any missed deadline. The latter allows such a test to be repeated in case of failures.
- Start with functional tests first, thereby using the host (simulation) PC only. Thereafter map the software for the system under test onto its intended hardware platform and start using the communication channels required for the execution of the test-cases. Repeat the functional tests executed earlier (simulation) on the host PC, thereby running the test-suite and test-harness on the host PC.
- If required for a proper execution of the test cases, or if required for a proper validation of the timely behaviour of the system under test, then map the functionality of the test-harness onto a Windows Kernel Mode Process using the Real-Time Windows Target toolset from MathWorks. Alternatively map the test-harness onto a dedicated hardware platform. In the latter case and explicit interface between the test-suite and the test-harness needs to be present. In the case that the Real-Time Windows Target toolset is used, communication between the test-suite and the test-harness is supported by the External Mode communication channel.
- After following this approach, it shall be possible to execute all test cases.

This approach is illustrated in Figure 39, Figure 40, Figure 41, Figure 42, Figure 43 and in Figure 44.

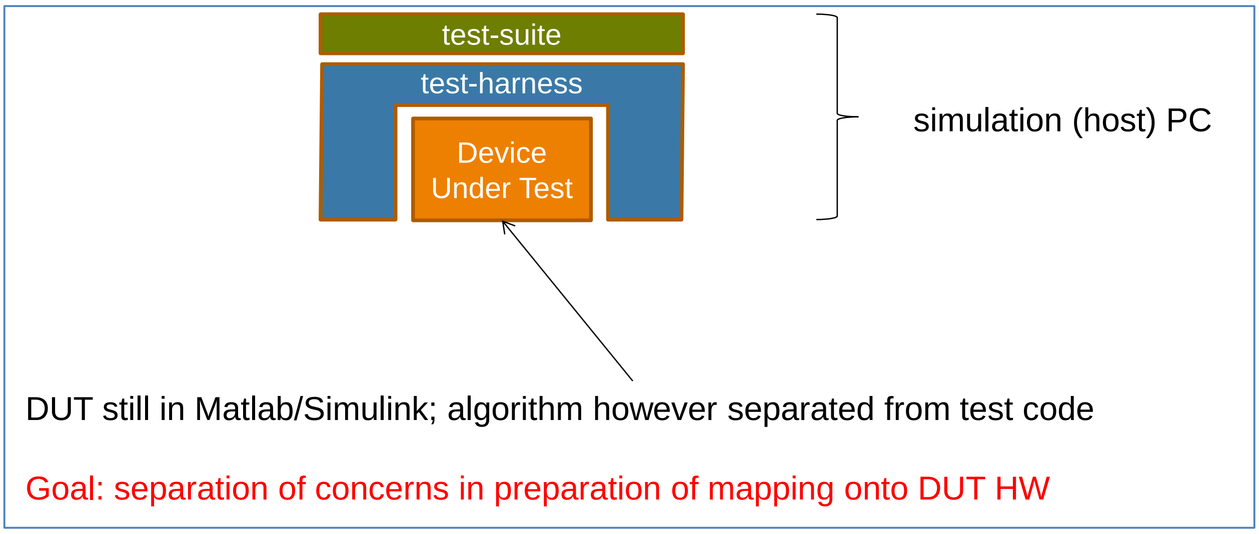

test-suite
test-harness
Device Under Test
simulation (host) PC
DUT still in Matlab/Simulink; algorithm however separated from test code
Goal: separation of concerns in preparation of mapping onto DUT HW

Figure 39: Functional validation (MIL)

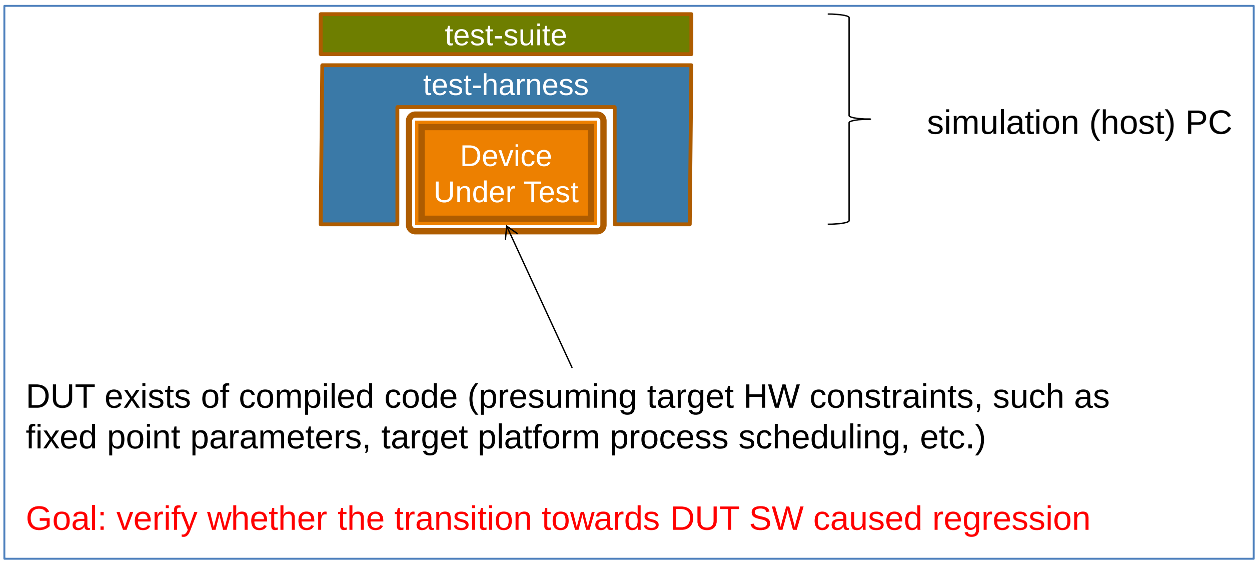

test-suite
test-harness
Device Under Test
simulation (host) PC
DUT exists of compiled code (presuming target HW constraints, such as fixed point parameters, target platform process scheduling, etc.)
Goal: verify whether the transition towards DUT SW caused regression

Figure 40: Functional validation (SIL)

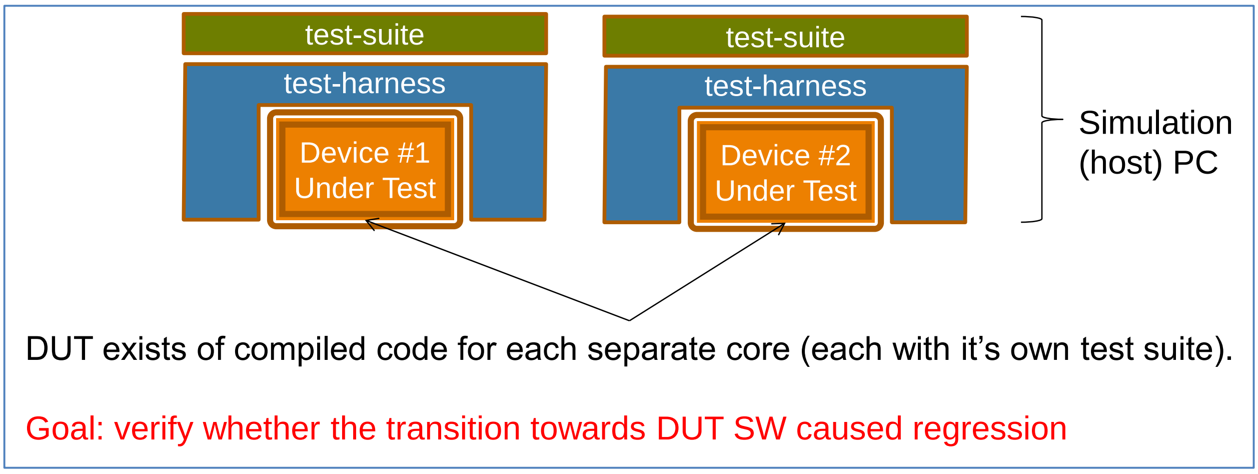

test-suite
test-harness
Device #1 Under Test
test-suite
test-harness
Device #2 Under Test
Simulation (host) PC
DUT exists of compiled code for each separate core (each with it's own test suite).
Goal: verify whether the transition towards DUT SW caused regression

Figure 41: Functional validation (SIL, multi-core, per core)

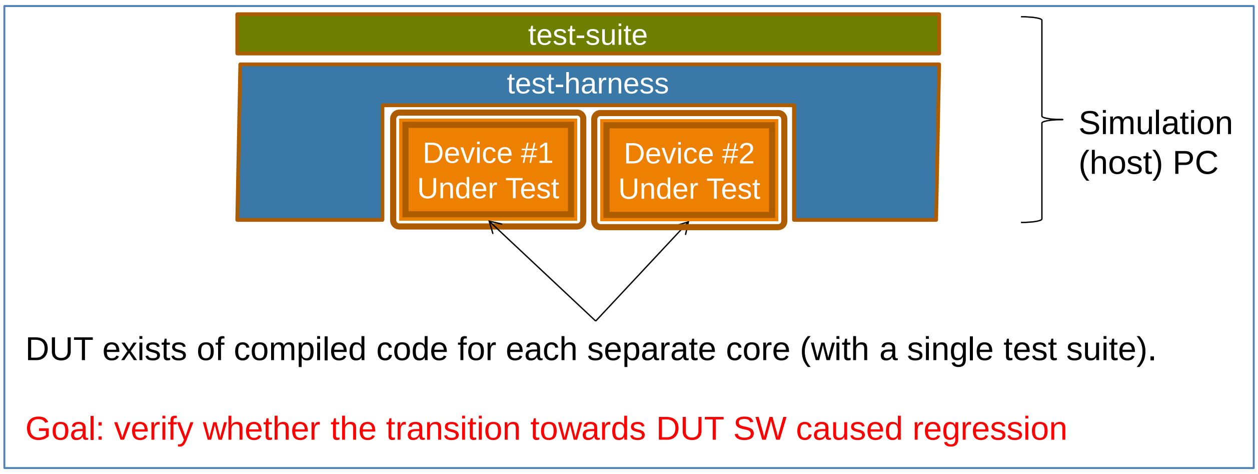

test-suite
test-harness
Device #1 Under Test
Device #2 Under Test
Simulation (host) PC
DUT exists of compiled code for each separate core (with a single test suite).
Goal: verify whether the transition towards DUT SW caused regression

Figure 42: Functional validation (SIL, multi-core, integrated)

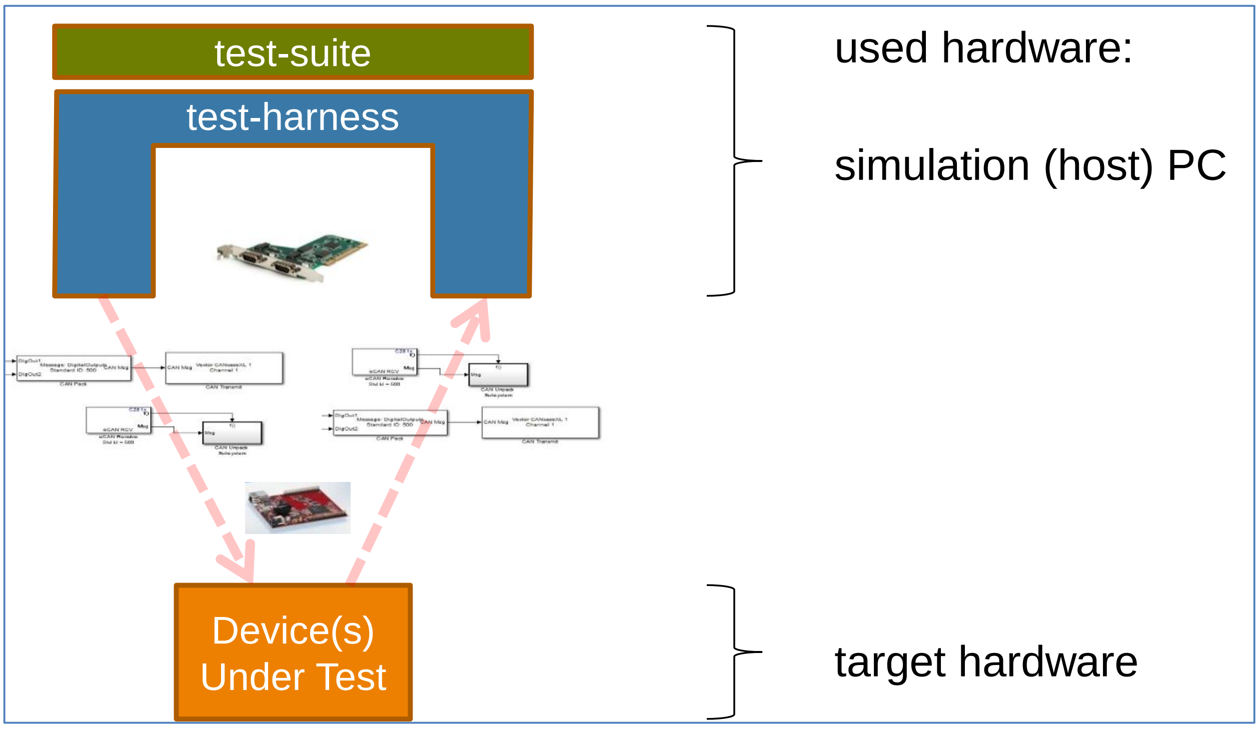


Figure 43: Functional validation (HIL)

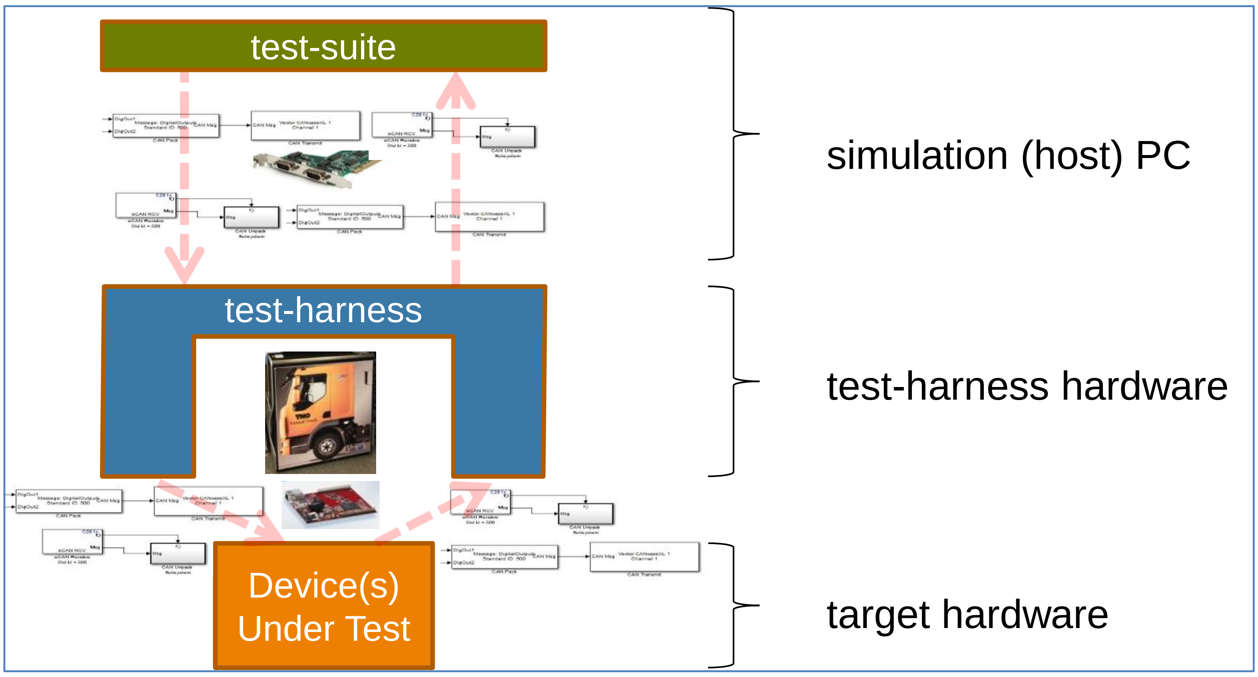


Figure 44: Functional validation (HIL, with real-time constraints)

7.1.6 ST7.3.6 UC_ Data exchange and communication strategies for complex multicore systems

7.1.6.1 Short description

As described in UC1.3_T1 to UC1.3_T6, multi-core platforms allow powerful solutions implementing complex simulation and control algorithms for next generation hybrid powertrains and e-Drive. However, this goes along with a strong increase in the complexity and required performance of data communication between these platforms and appropriate tools during development, calibration and diagnosis. Current in-vehicle network structures, its management and protocols used need to be improved to handle communication needs for multiple cores in a synchronous, low-latency way with enough throughputs. New methods are needed to manage e.g. consistency, variability, authenticity of data and its structure in such a highly parallelized system within the constraints of a vehicle's hardware.

Following topics shall be addressed in this task: new solutions for configuration, management, communication and verification of data exchange using automotive communication networks between multi-core based control units and development as well as test systems.

Relation to business needs: BN_T7.3_07, BN_T7.3_08
Relation to WPs: WP4

7.1.6.2 Spotlight outcomes

The design of the prototype consists of two components. On the one hand side the main processing component "Connectivity Manager", on the other hand side the "Connectivity Interface". Whereas there is only one instance of the Connectivity Manager running (singleton), every additional core is in control of its own instance of a Connectivity Interface. The main purpose of the Connectivity Interface (CI) is to establish a communication channel from the application, running on a random core, to the Connectivity Manager (CM), which is directly connected to the hardware. The inter-core communication therefore is achieved between CM and CI, which uses mailboxes for signaling. Data exchange per se is performed via a shared memory that is attached to both components.

The same inter-core communication mechanism will also be used for implementing proxies of shared devices. The rationale for this is that if a task may only access a shared device directly, then this task cannot be moved to another core without moving the device driver code to that other core as well. As other tasks may have a dependency on the to-be-moved device, these other tasks need to be moved as well. These dependencies will typically affect the freedom of task allocation and is therefore not desirable.

Whenever the interface of a device driver for a shared resource is defined, it will be possible to also generate the implementation code for a proxy device driver, which can run on one or more of the cores on which the actual device driver is not running[1]. This virtually allows the same context to be present for tasks running on any core. Moreover, these proxies support the mapping of safety critical tasks, together with the device (driver)s that have the same (derived) criticality, onto a core that is dedicated to run these safety critical applications. By using the inter-core communication for this purpose, criticality inversion and/or priority inversion issues can be avoided.

[1] Together with the device driver code, also code for a task that will provide the access point for all derived device driver proxies will be generated. This task will handle all messages originating from, and designated for, the proxies running on the other cores and will be running on the same core as the actual device driver.

7.2 High level requirements

Table 17: High level requirements for use case "Design and validation of next generation hybrid powertrain / E-Drive"

Req ID	Short description	Description	Rationale	BN rel	Sub-task rel
HL-REQ-WP07-007	Seamless tool chain - tool integration	Industrial tools such as e.g., Enterprise Architect, MKS, Matlab Simulink shall be integrated into a tool chain supporting seamless model-based systems / SW / safety engineering based on a semi-formal system description (e.g., EAST-ADL, SysML)	Tools integrated in the toolchain: ▪ Matlab/Simulink: modeling and software development ▪ Embedded Coder: c-code generation ▪ Hightec IDE: c-code development, os implementation, os configuration, middle implementation ▪ GCC compiler: c compiler ▪ PLS Debugger: debugging, tracing ▪ ICOS Independent CO-Simulation Platform: Co-simulation models, software, hardware with plant models	BN_T7.3_01	ST7.3.1, T7.3.5
HL-REQ-WP07-008	SW architectures - safety mechanisms	Scalable safety mechanisms shall be defined and implemented on the multicore platform to cover all automotive safety levels.	▪ Implementation of SoA and ORR approach ▪ M2C2 contract approach implemented for demonstrator ▪ Investigation of safety aspects with focus on data sharing strategies	BN_T7.3_02	ST7.3.2
HL-REQ-WP07-009	Safety case generation - automated compilation	The safety case information shall be automatically compiled based on existing information from product development.	▪ 2 evaluation scenarios executed for demonstrator platform ▪ Publications related to automated safety case generation	BN_T7.3_03	ST7.3.1, T7.3.3
HL-REQ-WP07-010	Simulation environment - multi-core emulation	A simulation environment shall be defined and implemented to provide emulation capabilities for multi-core platforms (highly parallel applications). This shall include following capabilities for an early evaluation of the SW design: a) Multi-rate simulation b) Scheduling analysis	▪ ICOS co-simulation platform used ▪ Evaluation of multi-core environment based on demonstrator platform	BN_T7.3_04	ST7.3.1, ST7.3.2, ST7.3.3
HL-REQ-WP07-011	Simulation environment - safety analysis	A simulation environment shall be specified and implemented in order to be able to provide support for the ISO26262 defined safety analysis of automotive safety critical applications.	▪ Implementation of SoA and ORR approach ▪ Runtime M2C2 contract approach implemented for demonstrator	BN_T7.3_05	ST7.3.1, ST7.3.4, ST7.3.5
HL-REQ-WP07-012	Network communication - data exchange	New approaches for the configuration and data exchange between the multicore control unit and developement / test systems over existing automotive communication channels /protocols shall be provided.	▪ implementation of the prototype done ▪ Long termtest runs executed (88h, 12 connecitons, 950 mil CAN messages, bandwidth utilization of 98%) ▪ connection manager for multi-core systems developed	BN_T7.3_06	ST7.3.3, ST7.3.4

7.3 Evaluation results

SFR: SW Application: Integration vehicle control unit / e-drive

The major objectives within this use case are the design and validation of several system architectures based on multi-core platforms in order to be able to handle this future trend in the automotive field. The use case addresses following main topics by applying an existing hybrid/electrical powertrain application:

- Integration of multi-core technology in existing control applications in order to provide more computing resources for (a) improvement of existing functions and (b) new functionalities.
- Combining two automotive controller units in one (EMCU + VCU = VEMCU): Electrical and functional integration of high dynamic e-drive system controls with time based vehicle control algorithm.

To achieve these objectives several technical aspects need to be analyzed for these new multi-core technologies:

- Integration of mixed criticality SW components on multi-cores
- Integration and handling of AUTOSAR for multi-core architectures
- Safety aspects for multi-cores
    - Program flow of software components executing on different cores
    - Data sharing strategies between OS tasks and cores
- Simulation concerns for parallel executing applications
- New networking solutions for the increased amount of ECU data

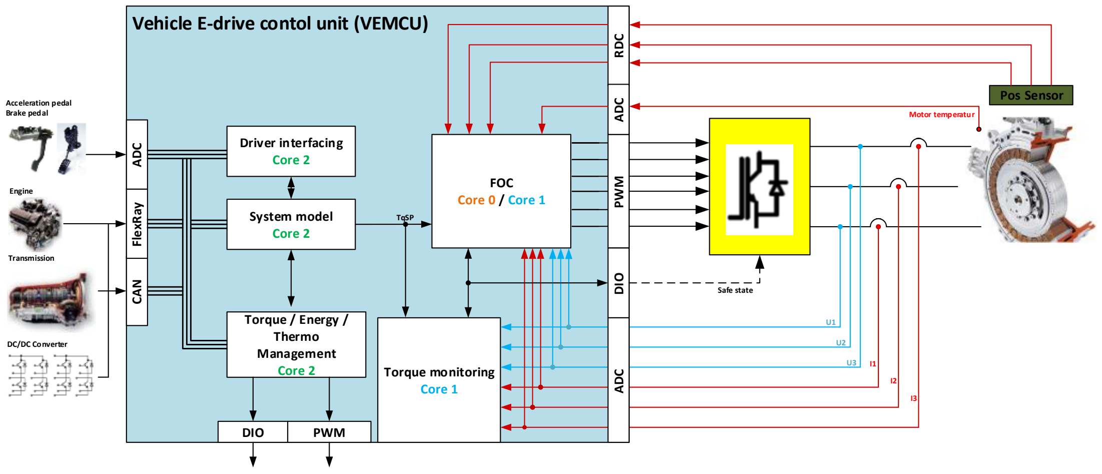


Figure 45: Overview of Vehicle Electric Motor Control Unit

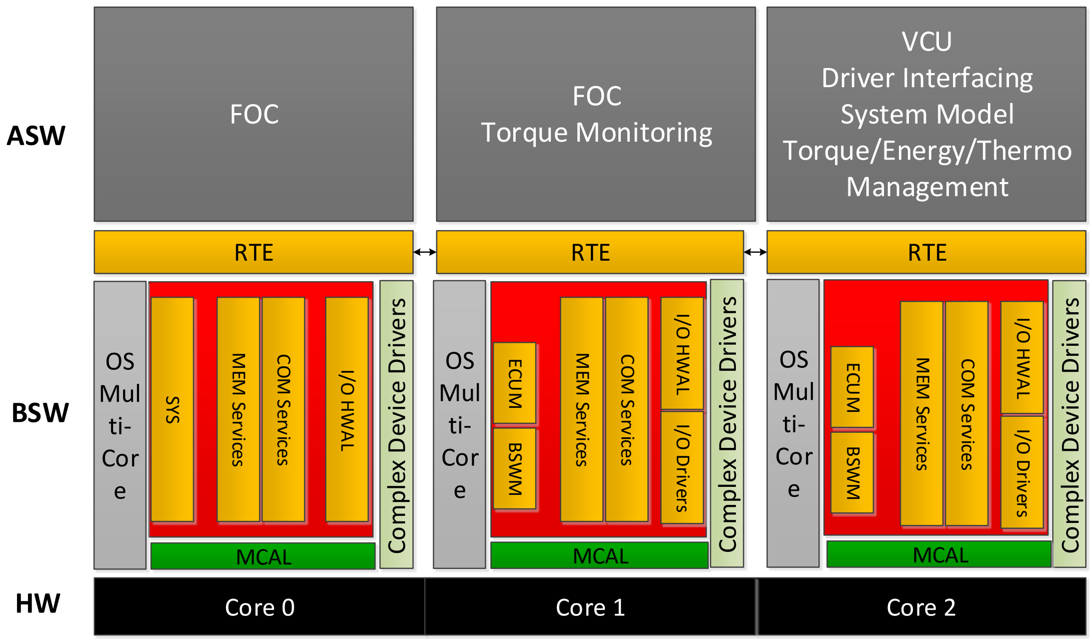


Figure 46: Overview of software distribution and interaction between cores

AVL SFR has so far intensively worked on the following:

- Investigation of different software architecture concepts on basic software level and application software level for the multicore approach
- Investigation of safety aspects with focus on data sharing strategies
- Adaption and development of basic software towards Multicore, e.g. MCAL, Complex Device Drivers, OS, RTE
- Adaption and development of application software towards Multicore: E-motor controls
- Electrical and functional integration of high dynamic e-motor controls on Aurix Multicore Eval Board
- Development of HiL Test set-up and multicore specific test cases
- HiL Test of e-motor controls

This leads to the following results:

- First version of BSW Multicore software architecture and guidelines
- AUTOSAR conform software
- E-Motor Controls is successfully integrated and operating on Aurix Multicore Eval Board
- The AUTOSAR conform software running is well integrated in the common demonstrator of all research partners
- Distribution of different e-motor control modules to different cores (core 0, core 1). First software version is available
- First version of tooling for the distribution of software modules available

Outlook:
AVL SFR is currently working on the distribution of the different e-motor control and VCU software modules to different cores (Core 0, Core1, Core 2). Focus will be the achievement of runtime optimization with respect to different mixed criticalities and safety aspects. Furthermore AVL will further adapt, develop and evaluate the tooling, meaning the distribution of software modules to the different cores shall be supported by tools.

TUW: ORR Integration

ORR can be connected to the system, e.g., on a separate platform, via a common communication network when the stated requirements are fulfilled. However, some design decisions make the integration easier. For instance, a publish/subscribe communication paradigm can decouple the monitoring and reconfiguration component from the application, i.e., application components are designed without considering structural adaptation and remain untouched during reconfiguration.

Regarding Execution Time

During this project the first prototype of ORR has been implemented on top of the robot operating system (ROS), which is a middleware enabling dynamic composability, common and reconfigurable communication. Unfortunately, ROS needs a significant amount of time to start or stop services and connect the services appropriately (mostly because of the flexible service management provided by ROS). However, the execution time of starting a substitute depends also on the platform and available resources (consumed also by other running services). Other middlewares may be used that fulfil the architectural requirements of ORR and (better) suit the target architecture increasing the performance of structural adaptation.
Once up and running, the substitute (or so-called transfer function) in our implementation has small execution time to other services, due to its simplicity. A substitute only consists of an interface for input and output, and the transfer function itself (typically expressing some law of physics). In ROS, services communicate directly, i.e., peer to peer, via TCP/IP messages without a management unit involved. The overhead of receiving information is linear with the number of inputs (a standard ROS interface for topics is used). Each transfer function has only one output, which overhead is therefore constant. Finally, the overall execution time also depends on the execution time of the transfer function, which can be evaluated beforehand.

AVL: Data exchange and communication strategies

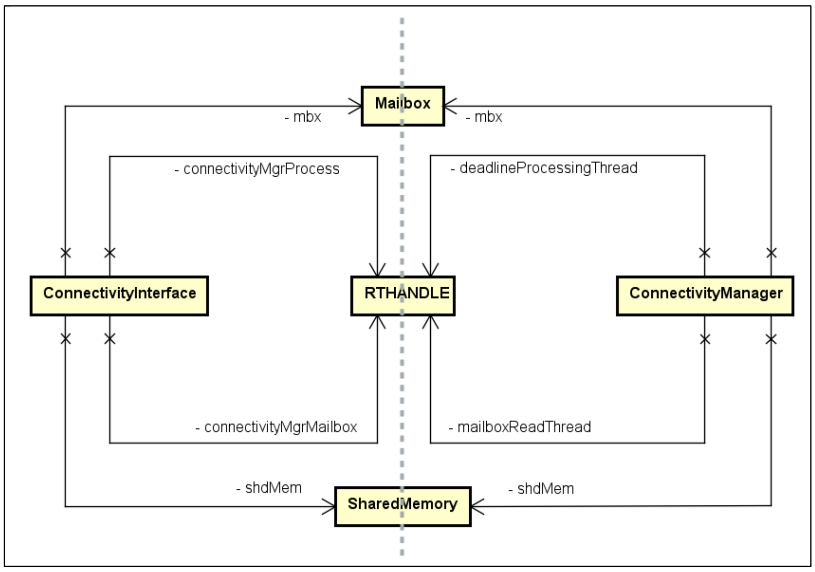


Figure 47: Inter-core communication between Connectivity Manager and Connectivity Interface.

The implementation of the prototype was done in C++ and runs as an independent .rta process on the real-time operating system INtime (version 4.20). As already stated, the prototype consists of two separate components, which are necessary to achieve an appropriate inter-core communication. As can be seen in Figure 47, both components, namely Connectivity Manager (CM) and Connectivity Interface (CI), hold a handle of the respective other component. CM and CI are both attached to the same shared memory and own a dedicated mailbox. CI has also a reference to the process of CM, so that the main processing component can be reached from a different core. Moreover, the CM additionally is in control of two threads, the “mailboxReadThread” and the “deadlineProcessingThread”. The “mailboxReadThread” listens for incoming mailbox messages while the “deadlineProcessingThread” processes all elements on a list until the list is empty. This concept follows the client-, worker-thread principle and enables efficient message processing without locking overhead. Whenever an application wants to register, write or read data, it calls the Connectivity Interface as a first step. The Connectivity Interface in turn notifies the

Connectivity Manager via mailbox message. With further information of the mailbox message the Connectivity Manager can perform the appropriate operation, in case of a “WRITE” request the CM will look up the according data with a specified offset within the shared memory. Thus, inter-core communication is performed via mailboxes while the actual data exchange is handled with the help of a shared memory. A novel message-scheduling algorithm is in charge of deciding which message is best to be sent next by the Connectivity Manager. The operating method of the scheduler is twofold. On the one hand, the scheduler operates on the earliest deadline first principle, which means that the message with the earliest deadline will be processed first (see Figure 48). Whenever the bandwidth of the bus (such as the CAN bus) decreases, due to retransmits or hardware errors, the Connectivity Manager might be incapable of sending messages before their according deadline. Thus, messages are delayed once a significant bottleneck on the bus occurs and not enough bandwidth remains. For this reason the scheduler is able to dynamically adjust its operating behavior. Whenever a delay occurred, which means that a deadline has been violated, the scheduler switches to “criticality first” method. At this scheduling

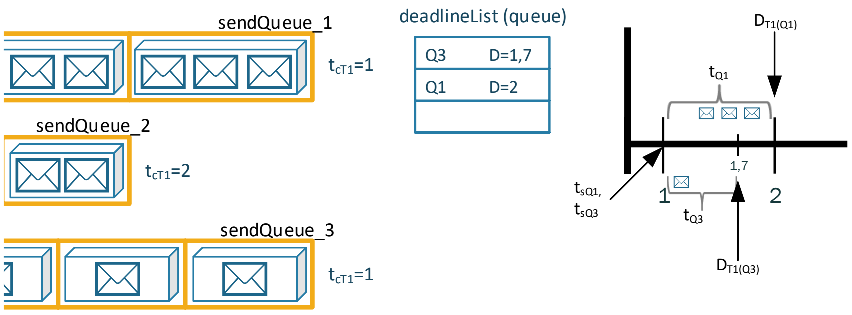
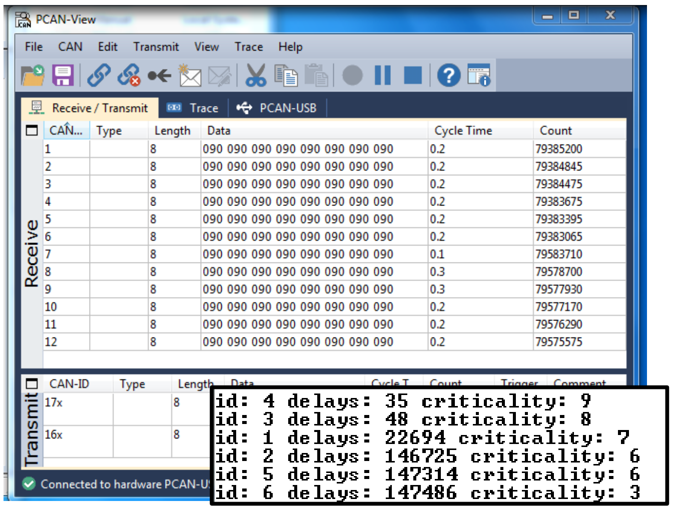


Figure 48: Earliest deadline first operating principle of the scheduler

behavior, information streams with a high criticality are prioritized, so that critical data, such as engine control data, stays at a real-time level. The system, including the scheduling algorithm, has been tested in various ways. The main criteria are as follows:

- System is stable in long term runs (> 48 hours)
- Dynamic scheduling is working
- Up to 12 connections can be handled at the same time
- Connections on varying cores perform equally
- A bandwidth utilization of 70 % can be reached without delay
- Faulty connections (varying amount of data) are handled correctly



CAN...	Type	Length	Data	Cycle Time	Count
1		8	090 090 090 090 090 090 090 090	0.2	79385200
2		8	090 090 090 090 090 090 090 090	0.2	79384845
3		8	090 090 090 090 090 090 090 090	0.2	79384475
4		8	090 090 090 090 090 090 090 090	0.2	79383675
5		8	090 090 090 090 090 090 090 090	0.2	79383395
6		8	090 090 090 090 090 090 090 090	0.2	79383065
7		8	090 090 090 090 090 090 090 090	0.1	79583710
8		8	090 090 090 090 090 090 090 090	0.3	79578700
9		8	090 090 090 090 090 090 090 090	0.3	79577930
10		8	090 090 090 090 090 090 090 090	0.2	79577170
11		8	090 090 090 090 090 090 090 090	0.2	79576290
12		8	090 090 090 090 090 090 090 090	0.2	79575575

CAN-ID	Type	Length	Data	Cycle T	Count	Trigger	Comment
17x		8					
16x		8					

```
id: 4 delays: 35 criticality: 9
id: 3 delays: 48 criticality: 8
id: 1 delays: 22694 criticality: 7
id: 2 delays: 146725 criticality: 6
id: 5 delays: 147314 criticality: 6
id: 6 delays: 147486 criticality: 3
```

Figure 49: 88 hours test run with additional bus load

After several test runs and applying various settings to the system, the following evaluation information is available.

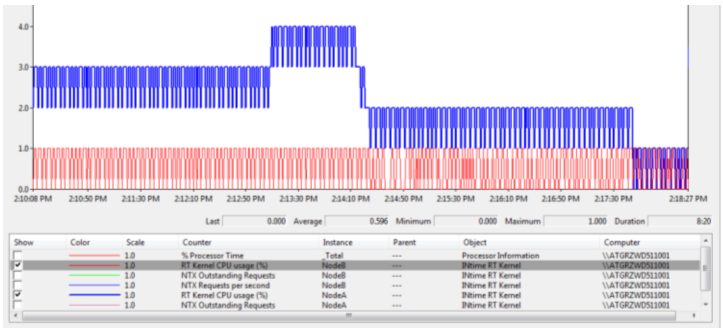
The system was successfully tested for its performance on a long term test run. Within 88 hours, 12 connections transmitted around **950 million CAN messages**, which were captured by the PCAN-View software (Figure 49). This result demonstrates on the one hand side, that the system is able to handle **long term runs** without failing. Moreover, it stays in a stable condition while managing **12 connections** simultaneously.

In another test scenario it has been proven that the system is capable of performing at a CAN bus **bandwidth utilization of 98%** without reporting any delays. Also the constraint of **faulty connections** has been tested and it turned out, that if a connection sends the double amount (compared to the ore-registered amount) of data sporadically, it does not influence the system, as long as there is enough bandwidth left to handle the load. For the system, this faulty behavior is detected just as an additional information stream. Finally yet importantly, the scheduling algorithm was tested in a vast of test runs. As it can be seen in Figure 49, the scheduling algorithm prioritizes critical information streams so that less critical information streams are delayed in favor of critical information streams if required.

During the test runs, the RT Kernel CPU usage was observed in order to determine the additional processor load. This observation indicates that CPU usage increases up to 6-8 % during stress phases. The peak of 8 % was reached while messages have been queued, due to a bottleneck, on the core that hosted the Connectivity Manager, Connectivity Interface, A2te Core Server and the jcan driver. An example of such a stress phase can be seen in Figure 50.

Concluding, the following results were achieved:

- Prioritization of information streams across multiple cores
- Bandwidth utilization up to 98 %
- Stable system in long term runs (88 hours)
- Handling multiple information streams simultaneously
- Minor increase of RT Kernel CPU usage (6-8 %)



Last 0.000 Average 0.596 Minimum 0.000 Maximum 1.000 Duration 8:20

Show	Color	Scale	Counter	Instance	Parent	Object	Computer
☐	🟥	1.0	% Processor Time	_Total	---	Processor Information	\\ATGRZWD511001
✓	🟥	1.0	RT Kernel CPU usage (%)	NodeB	---	INtime RT Kernel	\\ATGRZWD511001
☐	🟩	1.0	NTX Outstanding Requests	NodeB	---	INtime RT Kernel	\\ATGRZWD511001
☐	🟦	1.0	NTX Requests per second	NodeB	---	INtime RT Kernel	\\ATGRZWD511001
✓	🟦	1.0	RT Kernel CPU usage (%)	NodeA	---	INtime RT Kernel	\\ATGRZWD511001
☐	🟪	1.0	NTX Outstanding Requests	NodeA	---	INtime RT Kernel	\\ATGRZWD511001

Figure 50: Minor increase of RT Kernel CPU usage during stress phase of Connectivity Manager

7.4 Exploitation

The predicted strongly increasing market penetration of Advanced Driver Assistant Systems (ADAS) and (partially) automated vehicles is foreseen to triple to € 122,6 billion by 2021. Connected car developments will thus center on several functional areas, such as: (1) autonomous driving (including e.g. self-parking cars), (2) safety functionalities (such as danger warning signals and emergency call functions), (3) driver’s health and competence monitoring, (4) vehicle management (including remote control of car features), and (5) mobility management (more fuel-efficient driving and guidance on economical traffic ja The predicted strongly increasing market penetration of Advanced Driver Assistant Systems (ADAS) and (partially) automated vehicles is foreseen to triple to € 122,6 billion by 2021. Connected car developments will thus center on several functional areas, such as: (1) autonomous driving (including e.g. self-parking cars), (2) safety functionalities (such as danger warning signals and emergency call functions), (3) driver's health and competence monitoring, (4) vehicle management (including remote control of car features), and (5) mobility management (more fuel-efficient driving and guidance on economical traffic jam avoidance). Thus autonomous driving has the potential to transform mobility and traditional automotive industry.

To cope with the novel challenges of automated driving functions several approaches in the field of connected power train platforms, safety and security engineering framework and simulation and validation of distributed and connected automotive CPS have been analyzed.

These outcomes of the LL 7.3 will be used as input to upcoming research project. The approaches and the prototype architecture concepts will be handed over and reused for future projects dealing with novel embedded E/E architectures.

The first implemenations of SoA approaches (WP1) and runtime certification activities (WP6) forms a knowledge base for other national project proposal and the knowledge gained in EMC² has significantly raised the knowledge base of the involved partners and engineers.

The WEFACT, FMVEA, and PROSSURANCE tools and M2C2 methodology will be taken over for further internal evaluation and validation.
The Connectivity Manager (CM) and Connectivity Interface (CI) approach is currently further enhanced and refined for higher TLR to be implemented in future testbed applications.

m avoidance). Thus autonomous driving has the potential to transform mobility and traditional automotive industry.

To cope with the novel challenges of automated driving functions several approaches in the field of connected power train platforms, safety and security engineering framework and simulation and validation of distributed and connected automotive CPS have been analyzed.

These outcomes of the LL 7.3 will be used as input to upcoming research project. The approaches and the prototype architecture concepts will be handed over and reused for future projects dealing with novel embedded E/E architectures.

The first implemenations of SoA approaches (WP1) and runtime certification activities (WP6) forms a knowledge base for other national project proposal and the knowledge gained in EMC² has significantly raised the knowledge base of the involved partners and engineers.

The WEFACT, FMVEA, and PROSSURANCE tools and M2C2 methodology will be taken over for further internal evaluation and validation.
The Connectivity Manager (CM) and Connectivity Interface (CI) approach is currently further enhanced and refined for higher TLR to be implemented in future testbed applications.

8. Appendix 4: T7.4 UC Modelling and functional safety analysis of an architecture for ACC system

8.1 KPIs

The current design of automotive safety critical systems shall be supported by a suitable modelling methodology that could allow the conformity to ISO 26262 standard.
The high level requirements associated to the 7.4 use case are reflecting the needs of the proposed methodology development and associated objective and KPI, that are in relation to the deployment of a safety mixed criticality system project according to ISO 26262 standard.
Objective - OBJ41 Automotive: development of the specifications of a safe and trustworthy ACC (Adaptive Cruise Control) system architecture. The objective has been achieved through the analysis and the modelling of ACC system architecture for advanced vehicles from a functional safety perspective, considering the implication of the ISO 26262 standard and the related verification at concept level. This outcome will enable a sustainable design of future product oriented solutions in this field.
KPI - KPI33: Trustworthiness of ACC system architecture. Make effective the trustworthiness and, finally, the safeness specifications at concept level of ACC system architecture (starting from the state of the art prototype, the 100% of possible architecture malfunctions and related hazards has to be analysed in the corresponding operational situations and at least more than 75% of the necessary safety requirements at concept level have to be defined).

8.2 Sub task descriptions

The general aim of the use case was to develop a methodology for modelling by semi-formal language the safety critical systems and their safety requirements, in order to produce a virtual concept representing the architectural structure and the outcomes of the functional safety analysis, in a potential semi-automated way to be suitable for a future conformity assessment more reliable/robust efficient and less cost/time consuming.
The focus has been centred on the ACC system that will be considered and modelled from the functional safety perspective, in the context of the semi-formal language SysML, as provided by Enterprise Architect framework. Models for this aim have been defined and designed with their relationships in the context of a methodological approach conformant to the ISO 26262 standard process.

8.2.1 **ST7.4.1 UC_Requirements and specification of an ACC system architecture for advanced vehicles**

The ACC system is a complex system dedicated to the driver’s guiding assistance, in order to increase his comfort in maintaining the cruise speed and distance from forward vehicles. The aspects of the guidance management covered by an ACC device, consequently, heavily imply the functional safety of the vehicle and, in perspective, its security protection. Therefore such system is developed considering into its design the implications coming from ISO 26262 standard process prescriptions (functional aspects and application scenarios analysis for developing the subsequent hazards identification and classification).
The system functional requirements, its operating modes and significant operating scenarios were defined in an Office environment (Word/Excel), with dedicated templates, according to internal best practice outlining the ISO 26262 Item Definition, together with a system block diagram and state diagram representing the preliminary architecture.
In this best practice environment, as summarized in Figure 51, the system behavior shortfalls have been resumed in a list of malfunctioning behaviors that have been supposed to be the causes of hazards within the specific operational situations considered, according to the known operating modes and scenarios. These elements entered in the subsequent Hazard Analysis and Risk Assessment (HARA), as prescribed by ISO 26262 standard, for constituting the hazardous events, that have been classified through the

parameters Controllability [C], Severity [S] and Exposure [E]; then, for each of them, the followings have been determined: an ASIL (Automotive Safety Integrity Level) and a Safety Goal, this last finalized to the avoidance of the corresponding hazard into the hazardous event.

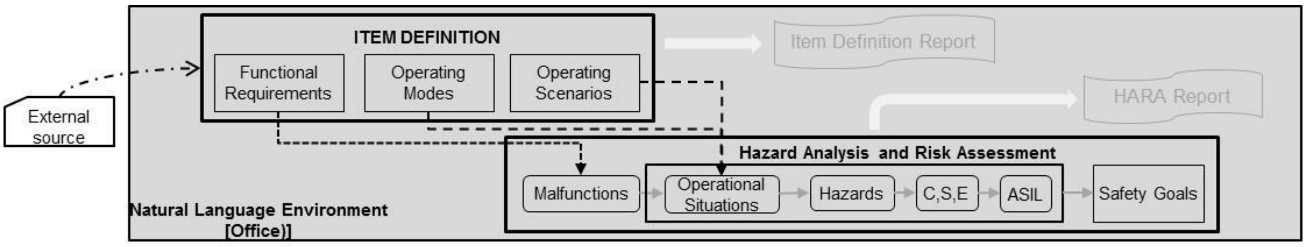


Figure 51: Best practice for Item Definition and HARA deployments (ISO 26262)

From the determined Safety Goals the functional safety requirements have been derived and integrated with the functional requirements in the preliminary architecture as from the previous Item Definition. The developed methodology has the aim of improving and completing the best practice for managing in an assisted/semi-automated and traceable way the safety requirements derived from the HARA.

The methodology, developed in the following tasks, covers the Safety Requirements Chain management, interfacing the inputs from Word documentation and Excel tables, and it is also applied to the part of the process related to the Item Definition and, partially, to HARA, as illustrated into the Figure 51. The safety requirements have been linked to the modelled Functional Architecture elements, in order to satisfy the ISO 26726 standard prescription for the allocation of the safety requirements to the system architecture elements, and the HARA elements were modelled to be the inputs for the Safety Goals at the top of the safety requirements chain (Figure 52 and Figure 53).

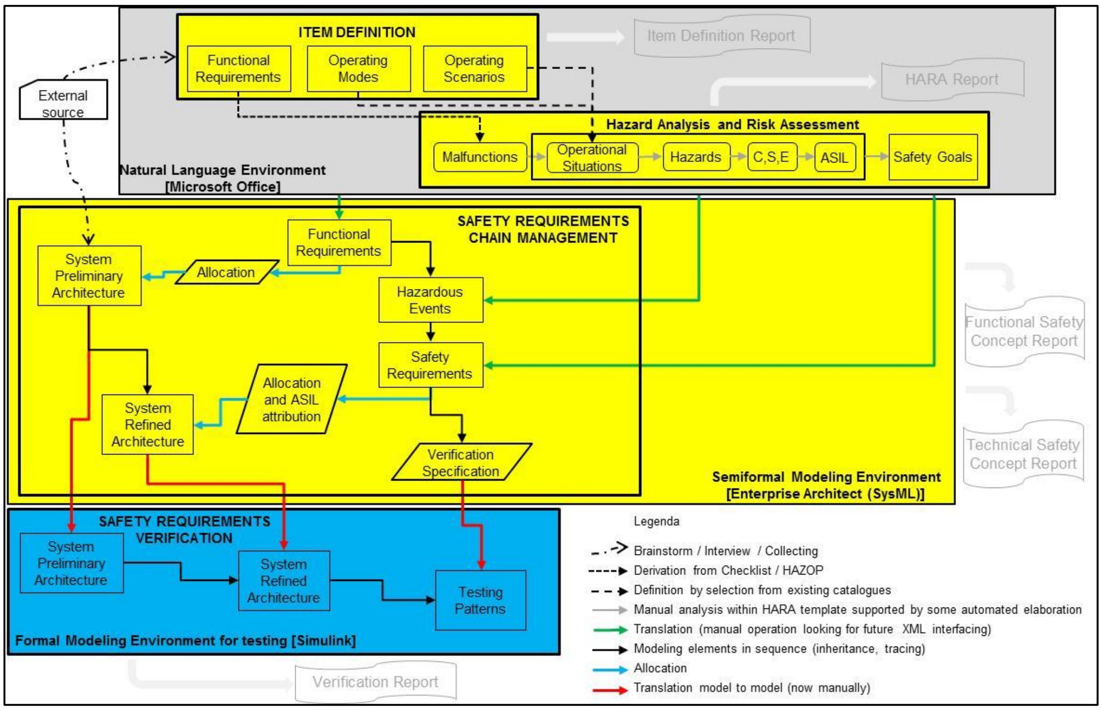

ITEM DEFINITION
Functional Requirements
Operating Modes
Operating Scenarios
Item Definition Report
HARA Report
External source
Hazard Analysis and Risk Assessment
Malfunctions
Operational Situations
Hazards
C,S,E
ASIL
Safety Goals
Natural Language Environment [Microsoft Office]
SAFETY REQUIREMENTS CHAIN MANAGEMENT
Functional Requirements
System Preliminary Architecture
Allocation
Hazardous Events
Safety Requirements
Allocation and ASIL attribution
System Refined Architecture
Verification Specification
Functional Safety Concept Report
Technical Safety Concept Report
Semiformal Modeling Environment [Enterprise Architect (SysML)]
SAFETY REQUIREMENTS VERIFICATION
System Preliminary Architecture
System Refined Architecture
Testing Patterns
Formal Modeling Environment for testing [Simulink]
Verification Report
Legenda
Brainstorm / Interview / Collecting
Derivation from Checklist / HAZOP
Definition by selection from existing catalogues
Manual analysis within HARA template supported by some automated elaboration
Translation (manual operation looking for future XML interfacing)
Modeling elements in sequence (inheritance, tracing)
Allocation
Translation model to model (now manually)

Figure 52: Implemented methodology (yellow background boxes) with respect to the current best practice

A dedicated application of Simulink, finally, covers the logical verification of the requirements.

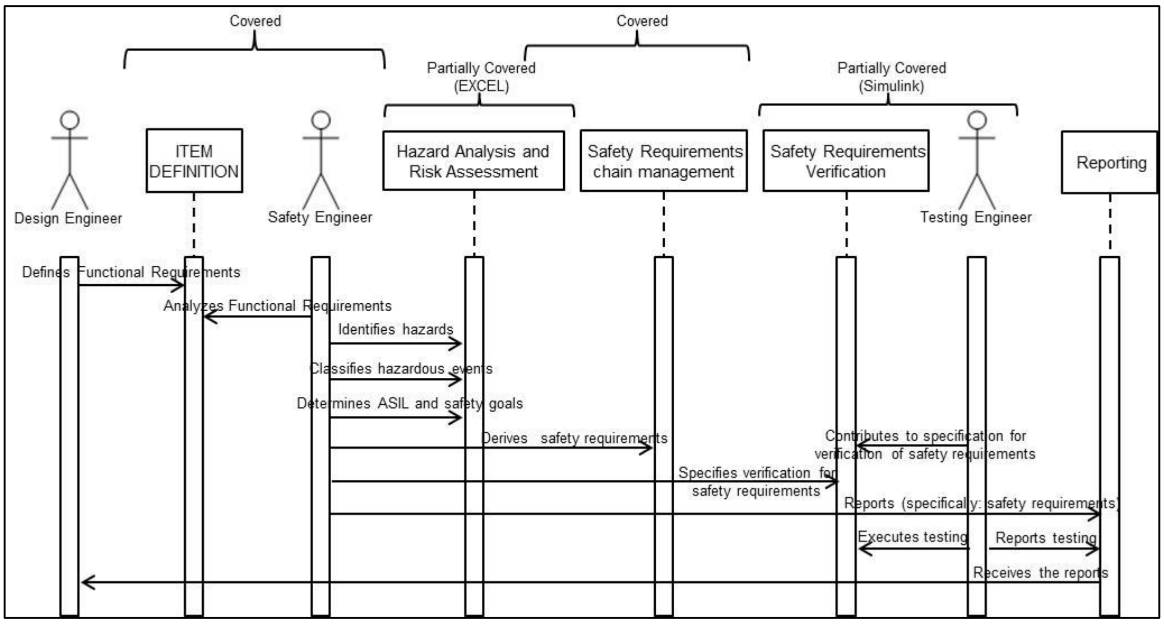

Covered
Covered
Partially Covered (EXCEL)
Partially Covered (Simulink)
Design Engineer
ITEM DEFINITION
Safety Engineer
Hazard Analysis and Risk Assessment
Safety Requirements chain management
Safety Requirements Verification
Testing Engineer
Reporting
Defines Functional Requirements
Analyzes Functional Requirements
Identifies hazards
Classifies hazardous events
Determines ASIL and safety goals
Derives safety requirements
Contributes to specification for verification of safety requirements
Specifies verification for safety requirements
Reports (specifically: safety requirements)
Executes testing
Reports testing
Receives the reports

Figure 53: Activities diagram and the aspects covered by the implemented methodology

8.2.2 ST7.4.2 UC_Analysis and modelling of an ACC system architecture for advanced vehicles

The process steps from the best practice, implemented by the previous task, were considered for the development of the modelling methodology, having the general scope to support the system development according to the ISO 26262 standard; the ACC system elements previously defined were the basis of the deployment of this scope. The detailed scope of this modelling methodology was to realize a specific meta-model/tool chain to support ISO 26262 prescriptions and deployment during system development.
From the available information elaborated about the ACC application system, in terms of functional requirements and operating modes, a modeling approach was explored and implemented in semi-formal language, for integrating both natural language descriptions and visual modeling, with the aim of constituting a meta-model and a future SW platform (tool-chain), tailored from an user perspective, for supporting the application of ISO 26262 process workflow to the system development. The basic idea if showed in Figure 54 and Figure 55.

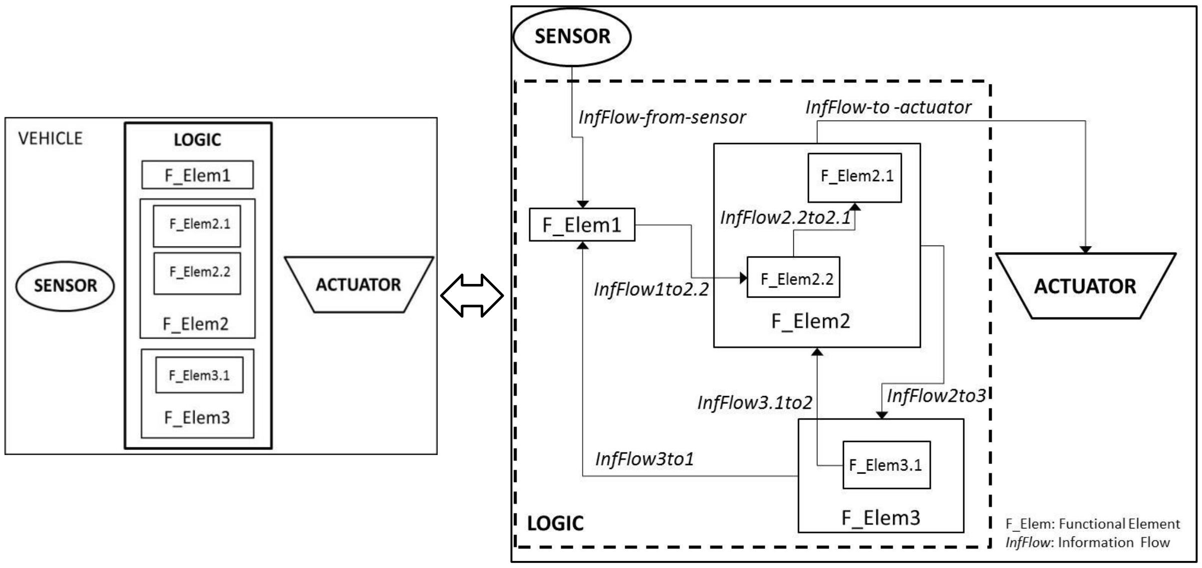


Figure 54: Basic semi-formal system architecture

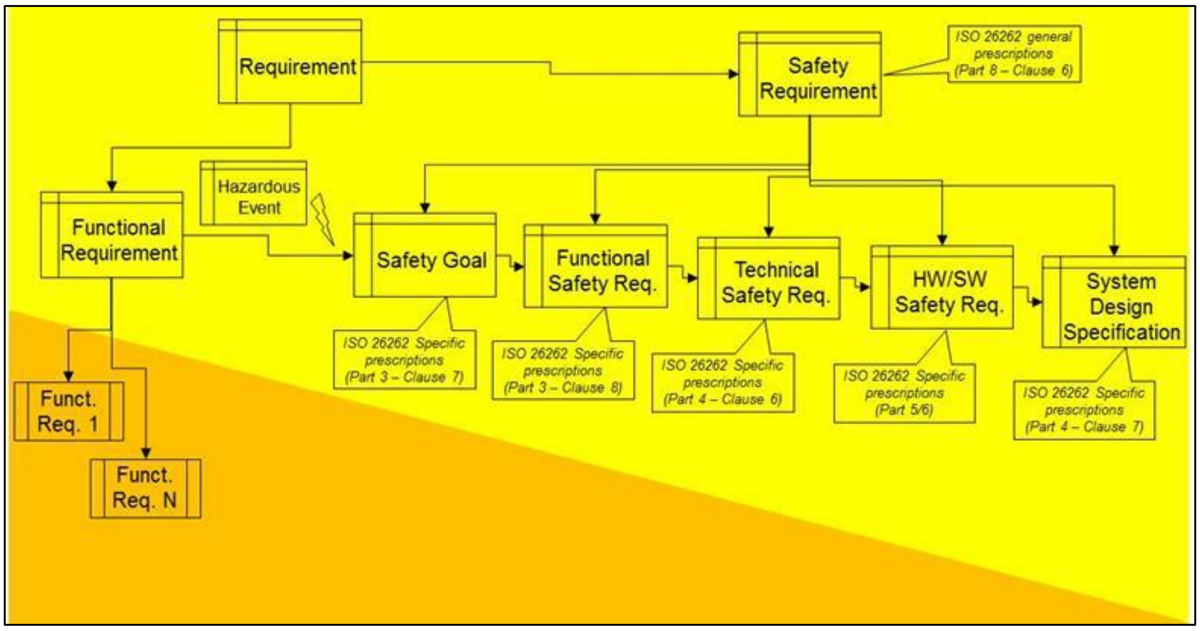

Requirement
Safety Requirement
ISO 26262 general prescriptions (Part 8 – Clause 6)
Hazardous Event
Functional Requirement
Safety Goal
Functional Safety Req.
Technical Safety Req.
HW/SW Safety Req.
System Design Specification
ISO 26262 Specific prescriptions (Part 3 – Clause 7)
ISO 26262 Specific prescriptions (Part 3 – Clause 8)
ISO 26262 Specific prescriptions (Part 4 – Clause 6)
ISO 26262 Specific prescriptions (Part 5/6)
ISO 26262 Specific prescriptions (Part 4 – Clause 7)
Funct. Req. 1
Funct. Req. N

Figure 55: Basic semi-formal requirements

The functional elements and their links constitute the functional architecture of the system that is also the preliminary architecture ISO 26262 standard requires, as supporting information during the functional and technical safety concepts elaboration. This architecture is completed by sensor elements, that constitute the source of the inputs to the functional architecture, and actuator elements, that work according to the outputs from the system: the result is the functional block diagram of the system as presented in the

Figure 56, which implement the basic idea, as from Figure 54, for the ACC system application.

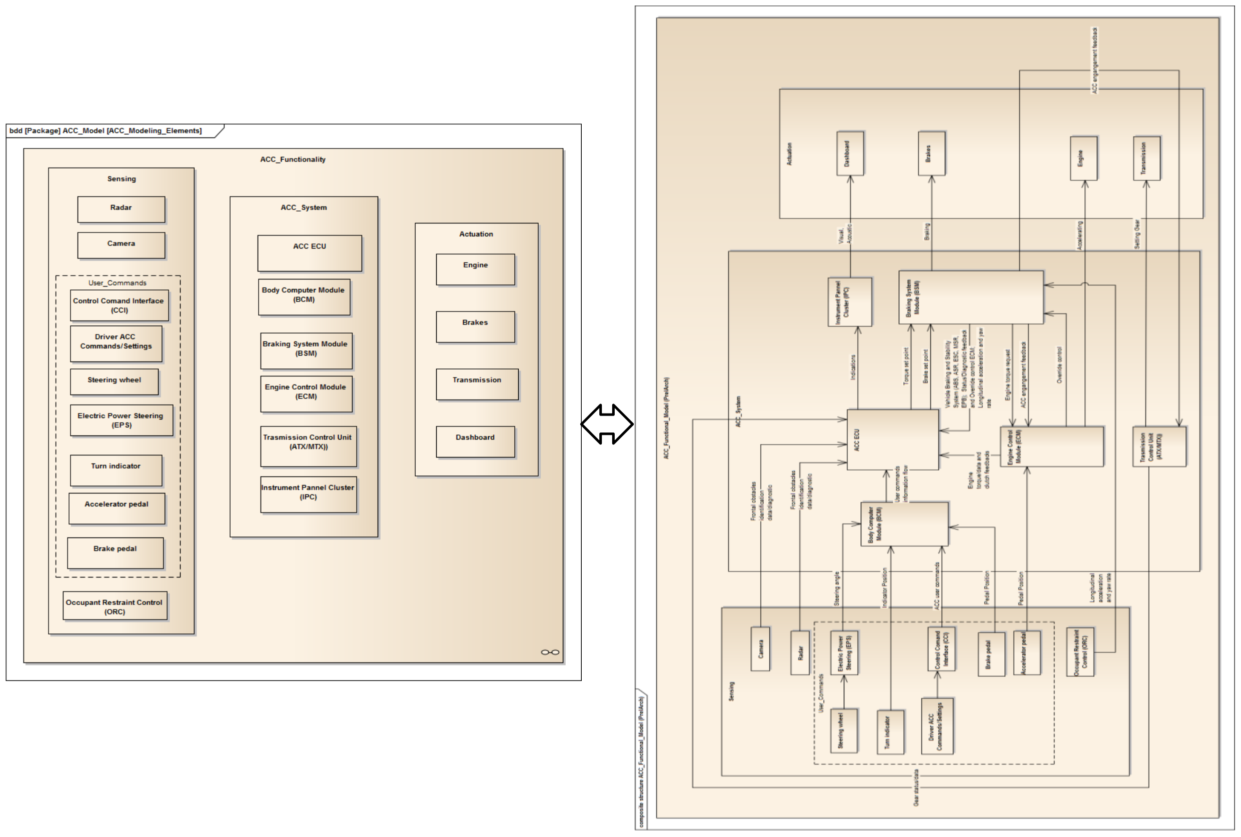

bdd [Package] ACC_Model [ACC_Modeling_Elements]
ACC_Functionality
Sensing
Radar
Camera
User_Commands
Control Comand Interface (CCI)
Driver ACC Commands/Settings
Steering wheel
Electric Power Steering (EPS)
Turn indicator
Accelerator pedal
Brake pedal
Occupant Restraint Control (ORC)
ACC_System
ACC ECU
Body Computer Module (BCM)
Braking System Module (BSM)
Engine Control Module (ECM)
Trasmission Control Unit (ATX/MTX)
Instrument Pannel Cluster (IPC)
Actuation
Engine
Brakes
Transmission
Dashboard

Figure 56: Semi-formal architecture for ACC system

This basis is deployed by the SysML/EA technic of the profiles definition/building, leading to the generation of suitable customized MDG (Model Driven Generation) technologies, for representing the architectural elements of the system (functional, design) and their relationships and for describing and representing the safety requirements with their attributes, hierarchy rules and relationships derived from the analysis.

The safety requirements modelling, anyway, required a more complex development of dedicated profiles (with related "stereotypes" and shapes), specific rules for inheritance of attributes, suitable automatic controls for links and allocation on the system architecture elements, which have been implemented in the following task, transforming the preliminary SysML modelling in a customized SysML semi-automated and assisted modelling.

8.2.3 ST7.4.3 UC_Functional safety implications according to ISO 26262 for the modelled architecture

According to the intended methodology, therefore, other than the functional elements and architecture, suitable artefacts where defined and implemented for representing all the other core elements of the functional safety process applied to the system: the Hazardous Events for the hazard analysis and risk assessment, the related Safety goals, the Functional and Technical Safety Requirements and their relationships to each other and to the system architecture for constituting the Safety Concept of the system.

In the safety requirements hierarchy chain management, from the point of view of the proposed methodology, it is important to support and, where possible, automatize the inheritance of the attributes (e.g. one for all the ASIL).

One of the main rules to be implemented was the establishment of a correct and controlled inheritance of the ASIL value from multiple sources (e.g. several Safety Goals with respect to the same Functional Safety Requirement). Then, it has been necessary to create a condition for automatizing the inheritance of a unique definition of ASIL attribute from all the Safety Goals and to associate to this condition an appropriate code that could implement the rules for inheritance/allocation control, maximum ASIL value calculation, ASIL decomposition guidance and traceability maintenance.

These rules have been implemented through dedicated scripts of code that have been integrated into the profile structure development associated to the specific artefacts related to the safety requirements modelling.
The Figure 57 shows the overall structure and the relationships of the artefacts developed for the safety requirements chain management.

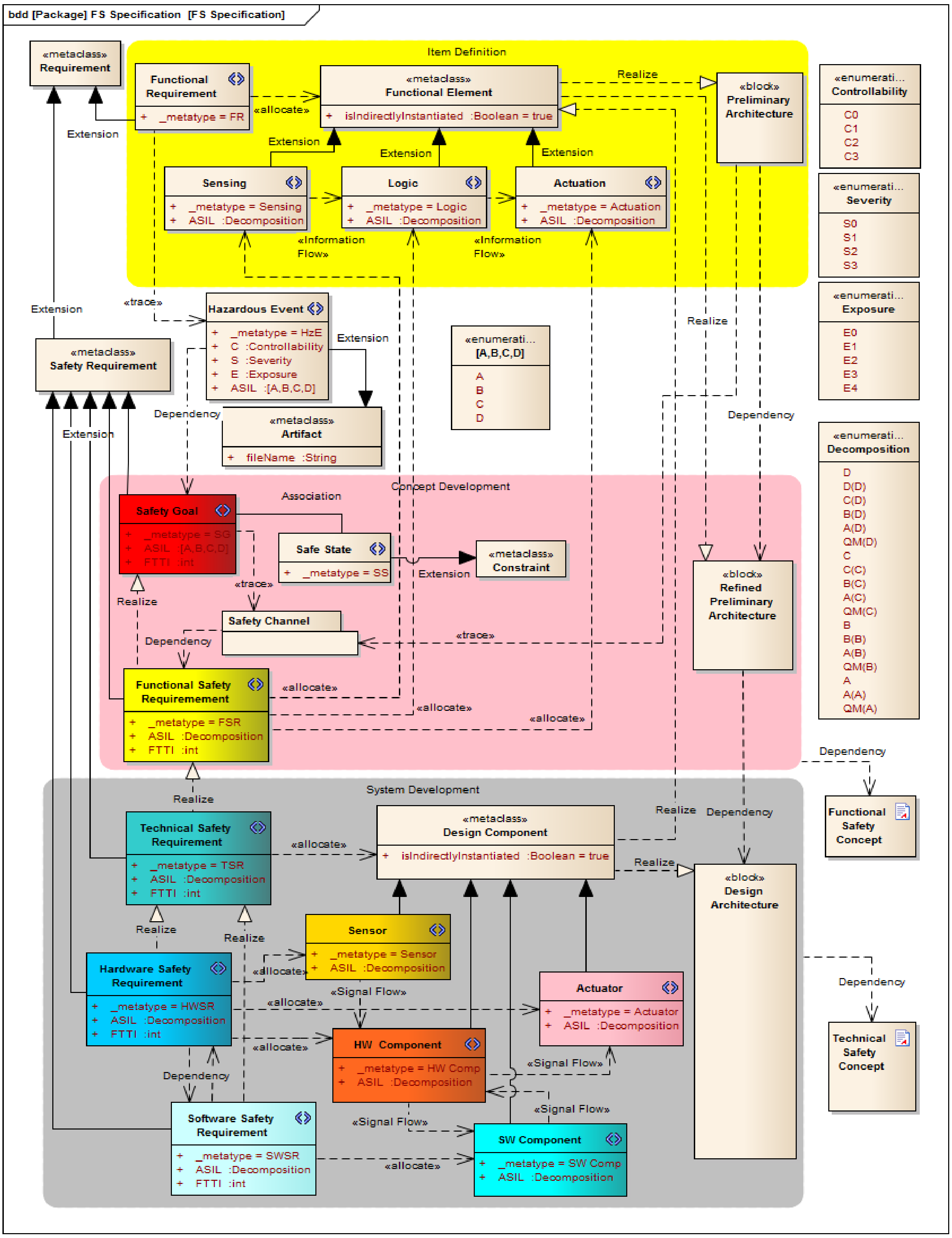
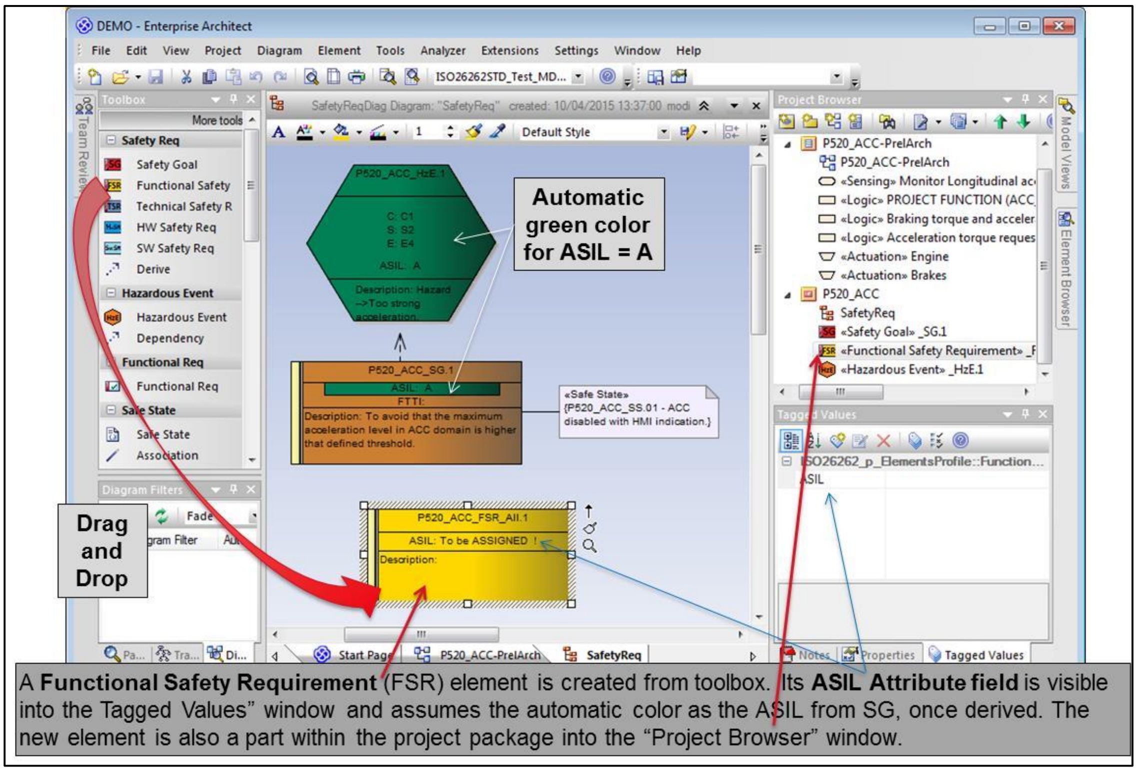


Figure 57: Semi-formal structure of the automated chain of the safety requirements (according to ISO 26262)

Finally, the safety requirements chain structure has been completed with a custom elaboration of Enterprise Architect framework allowing the use of the artefacts with simple drag and drop operations from dedicated palettes in dedicated windows, as showed in the example of Figure 58.


DEMO - Enterprise Architect
File Edit View Project Diagram Element Tools Analyzer Extensions Settings Window Help
Toolbox
Safety Req
Safety Goal
Functional Safety
Technical Safety R
HW Safety Req
SW Safety Req
Derive
Hazardous Event
Hazardous Event
Dependency
Functional Req
Functional Req
Safe State
Safe State
Association
Diagram Filters
Drag and Drop
SafetyReqDiag Diagram: "SafetyReq" created: 10/04/2015 13:37:00
P520_ACC_HzE.1
C: C1
S: S2
E: E4
ASIL: A
Description: Hazard -->Too strong acceleration
Automatic green color for ASIL = A
P520_ACC_SG.1
ASIL: A
FTTI:
Description: To avoid that the maximum acceleration level in ACC domain is higher that defined threshold.
«Safe State» {P520_ACC_SS.01 - ACC disabled with HMI indication.}
P520_ACC_FSR_All.1
ASIL: To be ASSIGNED !
Description:
Project Browser
P520_ACC-PrelArch
«Sensing» Monitor Longitudinal ac
«Logic» PROJECT FUNCTION (ACC
«Logic» Braking torque and acceler
«Logic» Acceleration torque reques
«Actuation» Engine
«Actuation» Brakes
P520_ACC
SafetyReq
«Safety Goal» _SG.1
«Functional Safety Requirement» _F
«Hazardous Event» _HzE.1
Tagged Values
ISO26262_p_ElementsProfile::Function...
ASIL
Start Page P520_ACC-PrelArch SafetyReq
Notes Properties Tagged Values

A **Functional Safety Requirement** (FSR) element is created from toolbox. Its **ASIL Attribute field** is visible into the Tagged Values" window and assumes the automatic color as the ASIL from SG, once derived. The new element is also a part within the project package into the "Project Browser" window.

Figure 58: Dedicated operations on safety requirements within the developed methodology

An example of a resulting safety requirement chain schema at functional level is showed in Figure 59.

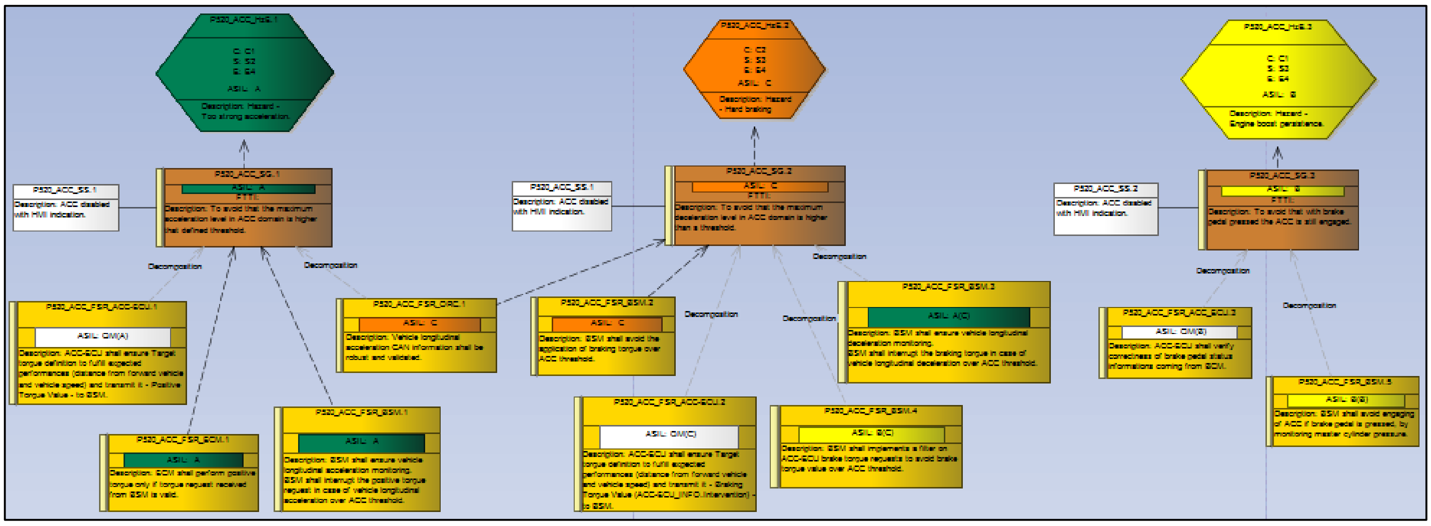


Figure 59: Safety requirements schema example

8.2.4 ST 7.4.4 UC_Evaluation

The Simulink section of the methodology is provided for supporting the logical verification of the safety requirements applied to the section of the system architecture related to each Safety Goal.
The elements of the architecture and their relationships are the same modelled in the Enterprise Architect. The Safety requirements are allocated visually on the elements and a dedicated Simulink model is applied to the relationships of the architecture elements for outlining the logical path of the failure propagation.
The sequence of the logic gates can be simulated for verifying the actual consistency and compliance of the safety concept, represented by the safety requirements allocated to the elements of the system architecture, with respect to the Safety Goals and its ability to mitigate or avoid the related hazardous events, according to the prescription of the ISO 26262 standard.
This kind of modelling is also a support for underlining the rationale of the ASIL value decomposition, if any occurred, which can applied only when the safety requirements are independent to each other with respect the failure propagation versus the Safety Goal: the specific "AND" gate represents this condition.
Figure 60 shows a graphical summary of modelling the safety requirements logical consistency/compliance verification, from the ACC system architecture section related to one of the Safety Goals, and, secondarily, representing the ASIL attribute value decomposition.

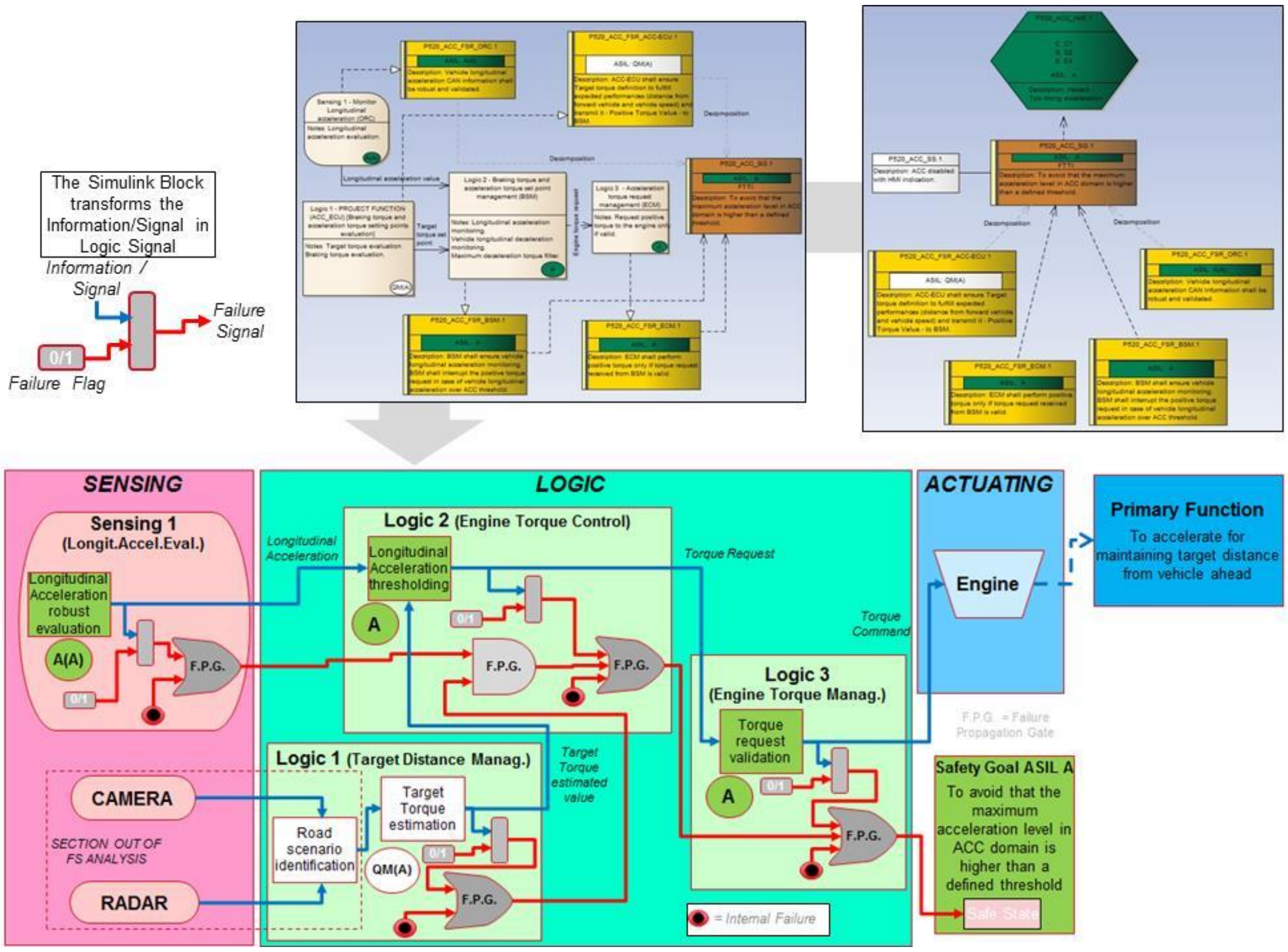

The Simulink Block transforms the Information/Signal in Logic Signal
Information / Signal
Failure Signal
0/1
Failure Flag
SENSING
Sensing 1 (Longit.Accel.Eval.)
Longitudinal Acceleration robust evaluation
A(A)
F.P.G.
CAMERA
RADAR
SECTION OUT OF FS ANALYSIS
LOGIC
Longitudinal Acceleration
Logic 2 (Engine Torque Control)
Longitudinal Acceleration thresholding
A
Torque Request
Torque Command
Logic 1 (Target Distance Manag.)
Road scenario identification
Target Torque estimation
QM(A)
Target Torque estimated value
Logic 3 (Engine Torque Manag.)
Torque request validation
= Internal Failure
ACTUATING
Engine
Primary Function
To accelerate for maintaining target distance from vehicle ahead
F.P.G. = Failure Propagation Gate
Safety Goal ASIL A
To avoid that the maximum acceleration level in ACC domain is higher than a defined threshold
Safe State
Decomposition

Figure 60: Modelling in Simulink to support safety requirements consistency and compliance

The verification of the safety requirements within their safety concept is on the scope for the evaluation of the methodological approach. But the modelling module in Simulink environment is not yet integrated into the overall framework of the methodology, requiring a more effort out of the limits of the current project. Anyway this remains a future challenge to be handled for a completely integrated methodological approach.

A semi-automatic reporting model has been finally produced, developed in a separated interface that reads the Enterprise Architect file/project (EAP) and writes down in a file Word, through a predefined template, the safety requirements chain (Figure 61) related to the system architecture diagrams.

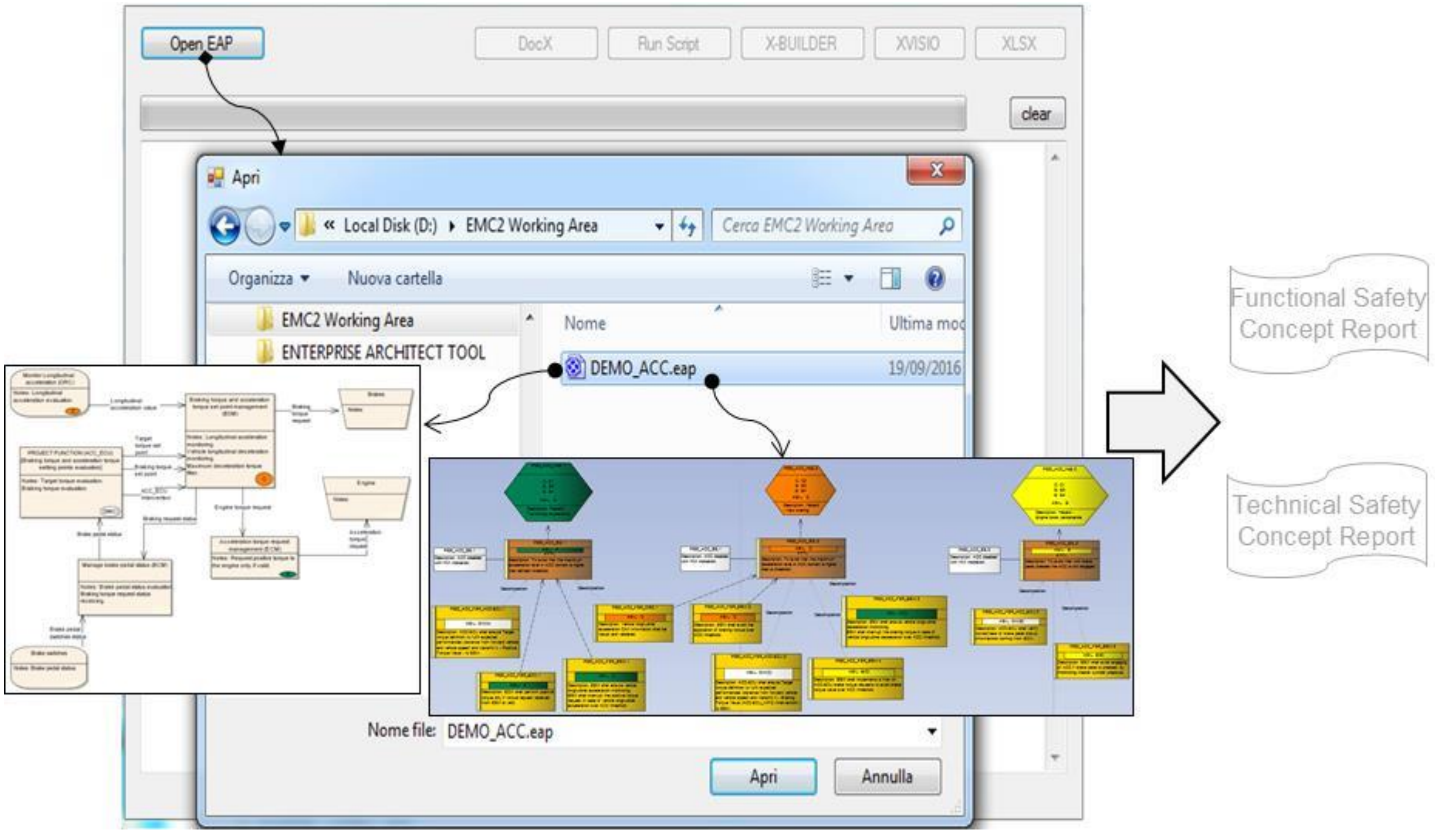


Figure 61: Reporting interface from safety requirements Enterprise Architect File/Project (EAP)

Finally, an example of a Safety Requirements chain until to the Technical Safety requirements is showed in the Figure 62. This is the input for the Technical Safety Concept Report output.

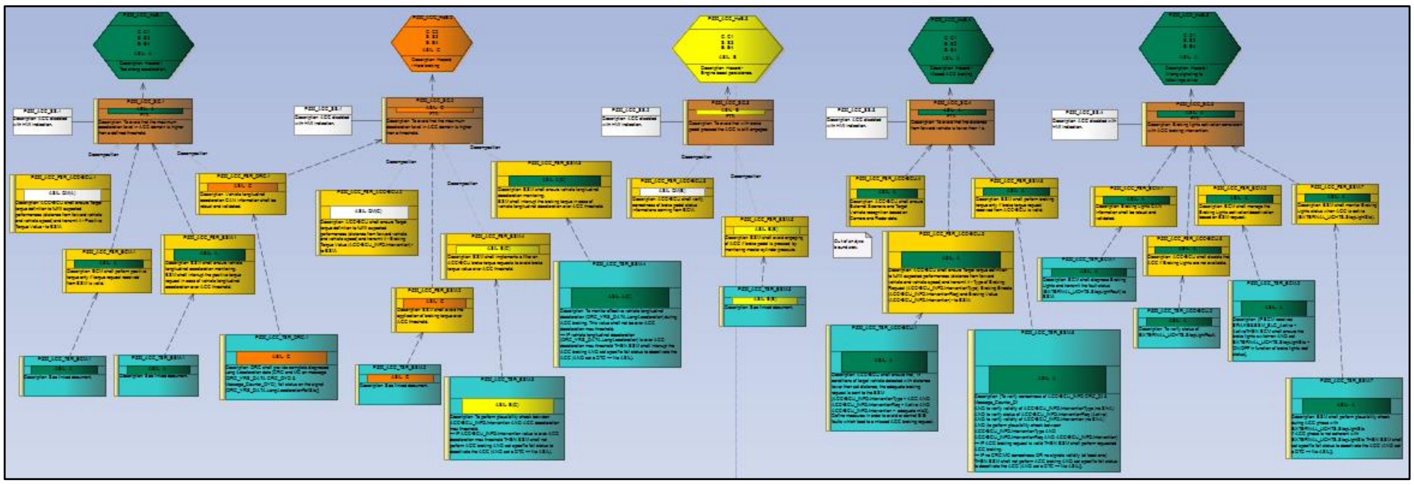


Figure 62: Example of Safety Requirements chain until to the technical safety requirements

8.3 High level requirements

The use case high level requirements reflect the needs of the proposed methodology development and associated KPI, that are related to the deployment of a safety mixed criticality system project according to ISO 26262 standard.
The high level requirements (reported in the following table) synthetize the required capabilities for system modelling, allocation of safety requirements and traceability of all the elements.
These needs have been evaluated through the application of the methodology to the ACC system and its safety requirements.

Table 18: High Level Requirements for UC7.4

Requirement ID	Short Description
HL-REQ-WP07-014	System architecture and functional requirements modeling in semi-formal language (e.g. UML/SysML) from Item Definition according to ISO 26262.
HL-REQ-WP07-015	Bi-directional translation and link between formal structure (e.g. Simulink) modeling and semi-formal (e.g. UML/SysML) modeling of the system architectures.
HL-REQ-WP07-016	ISO 26262 Hazard Analysis and Risk Assessment results translation/modeling in semi-formal language/environment (e.g. UML/SysML in Enterprise Architect).
HL-REQ-WP07-017	ISO 26262 safety requirements chain definition and derivation (starting from the safety goals) modeled in semi-formal language (e.g. UML /SysML).
HL-REQ-WP07-018	ISO 26262 safety requirements allocation to the system architecture (simulink and semi-formal language).
HL-REQ-WP07-019	ISO 26262 safety requirements internal compliance demonstration from lower level requirements to higher level requirements.
HL-REQ-WP07-020	ISO 26262 safety requirements verification tests automatic generation.
HL-REQ-WP07-021	ISO 26262 safety requirements traceability management.
HL-REQ-WP07-022	ISO 26262 safety case (before integration/validation) automatic generation from the results consequent to the above requirements.

8.4 Evaluation

The final evaluation criteria of the use case have been established in relation to the aims of the proposed methodology. In summary, they reflect the measure of the capability to support the description/modelling of the system and its safety requirements, effectively representing their connections and relationships and allowing the implementation of rules for the guidance and automation of the project through the steps prescribed by ISO 26262 standard safety cycle process.
The criteria, in summary, were as following:

1) Effectiveness of the representation/modelling of the architectural elements through suitable artifacts and representation/modelling of the elements' relationships through ad-hoc connectors/links.
2) Effectiveness of the representation/modelling of the hazardous events (hazards, operational situations safety relevant, malfunctioning behaviors, rank of classification in terms controllability, severity, exposure and consequent ASIL attribute value) and their links to safety goals.
3) More in details in relation for the safety requirements:
   a) Effectiveness of the representation/modelling through suitable artifacts of the various categories of safety requirements, starting from safety goals, with associated safe states, and taking into account their attributes (e.g. ASIL) and their hierarchical relationships.
   b) Effectiveness of managing the rules for ASIL inheritance/decomposition.
   c) Support to allocation of the modelled safety requirements to the system architecture elements and reciprocal traceable and graphical visibility.
4) For verification of safety requirements:

a) Support to representation/modelling of test patterns for the safety requirements, taking into account the link for the traceability of the results and their impact on the overall safety requirements chain: partially done in SysML/EA, to establish in a future methodological development the relationships towards the specific testing environments, such as Matlab/Simulink.
b) Effectiveness of modeling in Simulink context for supporting the verification of the consistency and compliance of the safety requirements developed in EA/SysML.

The effectiveness of all the foreseen capabilities, in general, should contribute to the self-reliance and robustness of all the information related to the functional safety analysis of the system and its related safety requirements definition, supporting the traceability throughout all the steps of the process and the guidance/control of the application of the rules for ASIL attribute values inheritance/decomposition. The ACC system application modelled with the deployed methodology confirmed the expected capabilities of the methodology in terms of system and safety requirements chain hierarchy representation / modelling and management. All the capabilities have been additionally increased by appropriate customized tool-boxes, supporting the user work.

In summary, the ACC use case assessment showed that the main praise of the proposed methodology is that the prescriptions from the ISO 26262 standard, about the generation and management of the safety requirements hierarchy, are supportable by a customized framework in semi-formal language. The framework facilitates the user work and increases the reliability of the job done, guaranteeing a control and an automation of the activity steps in a way that reduces to zero the possible errors in derivation, allocation and ASIL values inheritance/decomposition of the entire Safety Requirements chain, starting from the Hazard Analysis and Risk Assessment.

The point of the safety requirements verification/testing, unfortunately, has not yet been covered directly by the framework itself. Therefore it is not yet integrated on it, because SysML/EA framework and Simulink environment require for communicating a further interfacing development out of the limits of the current project. Then, the Simulink testing model application results in a separated job manually related to the SysML/EA artefacts.

8.5 Exploitation

The developed methodology is in progressive utilization within CRF architectural and design working groups. In the next future, an extension of it is foreseen within the other corresponding groups into the Product Development Departments of the regions EMEA (Europe) and NAFTA (USA) of the FCA main holding.

The kind of the methodology requires a minimum skill in managing graphical artefacts, but this job is well supported by the simplifications of the user interface and the operative menus, which characterize the chain tool realized, through the solutions implemented for handling the requirements and the related architectural elements in the friendliest way.

The methodology introduces a customized approach to SysML and promotes an internal increasing of competencies in requirements management, that summarizes the basic recommendations from INCOSE (International Council on Systems Engineering) about requirements and the more detailed prescriptions form ISO 26262 standard (Road Vehicles – Functional Safety – Parts 1 to 10) more specifically finalized to the system functional safety.

The maintenance of the chain tool, moreover, implies also the involvement of our ICT (Information and Communications Technology) Department as technological support. The use of the methodology, in perspective, will improve also the relationships with the suppliers allowing a simpler and clearer representation of the requirements and their relationships to the system under design.

9. Appendix 5: T7.5 UC Infotainment and eCall Application Multi-Critical Application

9.1 Relation between Business Needs and KPIs

The business needs behind this use case are described in the table below:

Table 19: Business needs for use case "Infotainment and eCall Application Multi-Critical Application"

No	Title	Description
BN_T7.5_01	Mixed criticality	To reduce certification costs, the system must support execution environments of different levels of safety integrity levels.
BN_T7.5_02	Multicore CPU	To address mixed criticality environments and the increasing concurrent tasks paradigm infotainment systems require, the system must allow concurrent mixed-criticality software execution.
BN_T7.5_03	Secure communication	Both critical and non-critical environments must be able to share data.
BN_T7.5_04	General purpose OSs	In order to allow an easier and faster development platform a general purpose OS shall be used for the infotainment GUI.
BN_T7.5_05	Real Time OS	Critical tasks require hard real time scheduling and shall be managed by a RTOS.
BN_T7.5_06	Infotainment	The system shall provide a graphical user interface with infotainment capabilities, e.g., multimedia, GPS navigation and internet connectivity.
BN_T7.5_07	System peripherals	The system should support touchscreen, 2D/3D HW accelerated GPU, 3 axis accelerometer, GPS, sound card and a GSM/3G modem.
BN_T7.5_08	eCall system	The system should provide the functionality to implement an eCall emergency system.
BN_T7.5_09	Hardware Assisted Virtualization	To improve performance and reduce development and certification effort the hardware shall support virtualization extensions, such as ARM's TrustZone.
BN_T7.5_10	Resource sharing	The system must allow for individual HW resources to be partitioned and/or shared between environments of distinct criticality
BN_T7.5_11	Resource criticality reassignment	The system must allow for a HW resource to be repurposed from a lower criticality environment to a higher on in case of an event occurred.

The relation between the KPIs and the Business Needs is shown in the table below:

Table 20: KPIs – Business Needs relation for use case "Infotainment and eCall Application Multi-Critical Application"

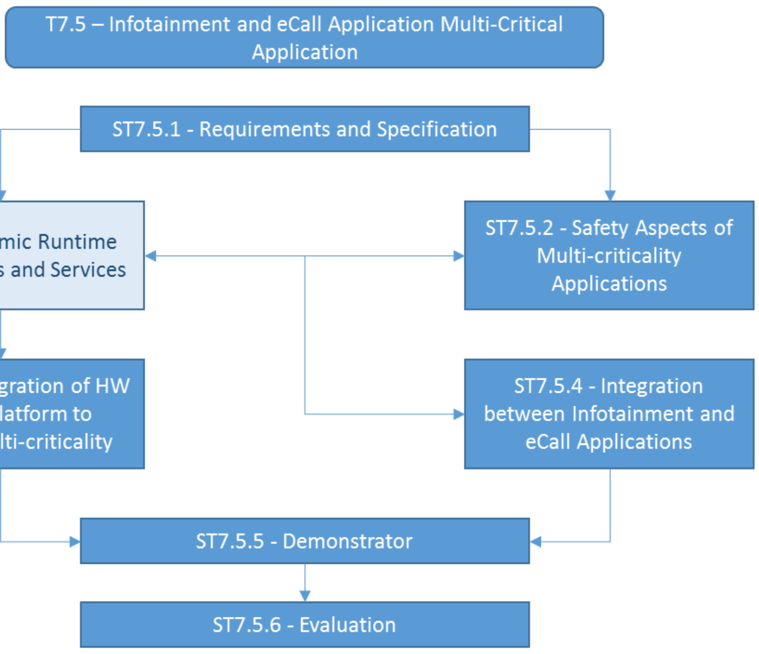
No	Title	Description	BN relation
KPI_T7.5_01	Mixed criticality tasks	The execution environment must support the deployment of tasks with at least two distinct levels of criticality	BN_T7.5_01
KPI_T7.5_02	Multicore tasks	The execution environment must support the deployment of concurrent multi-core applications.	BN_T7.5_02
KPI_T7.	Secure	A communication mechanism must be available for secure data transfer between	BN_T7.5_03

5_03	communication	two environments of distinct criticalities	
KPI_T7.5_04	Android	The Android Operating System shall be the runtime environment used as the Infotainment system	BN_T7.5_04 BN_T7.5_06
KPI_T7.5_05	FreeRTOS	The FreeRTOS shall be used as the runtime environment for th e RTOS	BN_T7.5_05
KPI_T7.5_06	Infotainment	The infotainment system shall use the Android applications for its multimedia purposes	BN_T7.5_04 BN_T7.5_06
KPI_T7.5_07	System devices	HW resources shall be managed by their assigned OS	BN_T7.5_07
KPI_T7.5_08	eCall system	An eCall system shall be developed as a critical real time task on the RTOS runtime environment	BN_T7.5_08
KPI_T7.5_09	Software Containment	The distinct levels of criticality shall be addressed using hardware assisted virtualization.	BN_T7.5_09
KPI_T7.5_10	Resource partitioning	The execution environment shall allow for space and time partitioning.	BN_T7.5_09 BN_T7.5_10 BN_T7.5_11
KPI_T7.5_11	Resource takeover	The execution environment shall support non-critical resource takeover by critical component initially determined as non-critical.	BN_T7.5_10 BN_T7.5_11
KPI_T7.5_12	Inter-core communication	The execution environment shall support an inter-core communication API and be able to send a signal between cores.	BN_T7.5_03
KPI_T7.5_13	Execution monitoring	The execution environment should at least support the monitoring of CPU utilization and idle time.	BN_T7.5_02
KPI_T7.5_14	Core affinity	The execution environment should allow binding a function to a specific core.	BN_T7.5_02

9.2 Sub task descriptions

In order to develop a multi-core multi-criticality demonstrator we've split into sub-tasks the various steps required to design, develop and integrate the technology used in the use case's demonstrator.

As described in Figure 63, the use case started with the collection of requirements in ST7.5.1 that propagated to both the use case sub-tasks and to the WP3 technology tasks. In ST7.5.2 the safety aspects were addressed throughout the duration of the use case in tight integration with T3.5. Technologies developed in WP3 were developed and integrated into the use case demonstrator within ST7.5.3 while the infotainment and safety-critical applications were handled in ST7.5.4. All this integration was finalized during ST7.5.5, where all the components were fully integrated to conclude the use case demonstrator, which is now reviewed and evaluated in ST7.5.6.


T7.5 – Infotainment and eCall Application Multi-Critical Application
ST7.5.1 - Requirements and Specification
WP3 – Dynamic Runtime Environments and Services
ST7.5.2 - Safety Aspects of Multi-criticality Applications
ST7.5.3 – Integration of HW and SW Platform to Support Multi-criticality
ST7.5.4 - Integration between Infotainment and eCall Applications
ST7.5.5 - Demonstrator
ST7.5.6 - Evaluation
Figure 63: Subtasks of T7.5 and interaction between WP7 and WP3.

9.2.1 ST7.5.1 Requirements and specification

In this first sub-task, all business needs (Table 19), high level and technology requirements were specified and contributed to T3.1 and WP7.
These influenced the use case demonstrator design and correspondent deliverables D7.11 and D7.12, as well technology tasks in WP3.
In the next tables are the use case's high level requirements and technology requirements as result from this sub-task.

Table 21: High Level Requirements

Requirement ID	**Title**	**Description**
HL-REQ-WP72-01	Mixed criticality tasks	The execution environment must support the deployment of tasks with at least two distinct levels of criticality
HL-REQ-WP72-02	Multicore tasks	The execution environment must support the deployment of concurrent multi-core applications.
HL-REQ-WP72-03	Secure communication	A communication mechanism must be available for secure data transfer between two environments of distinct criticalities
HL-REQ-WP72-04	Android	The Android Operating System shall be the runtime environment used as the Infotainment system
HL-REQ-WP72-05	RTOS	A real time operating system shall be used as the runtime environment for scheduling hard real time tasks.
HL-REQ-WP72-06	Infotainment	The infotainment system shall use the Android applications for its multimedia purposes
HL-REQ-WP72-07	System devices	HW resources shall be managed by their assigned OS
HL-REQ-WP72-08	eCall system	An eCall system shall be developed as a critical real time task on the RTOS runtime environment
HL-REQ-WP72-09	Software Containment	The distinct levels of criticality shall be addressed using hardware assisted virtualization.

HL-REQ-WP72-10	Resource partitioning	The execution environment shall allow for space and time partitioning.
HL-REQ-WP72-11	Resource takeover	The execution environment shall support non-critical resource takeover by critical component initially determined as non-critical.
HL-REQ-WP72-12	Inter-core communication	The execution environment shall support an inter-core communication API and be able to send a signal between cores.
HL-REQ-WP72-13	Execution monitoring	The execution environment should at least support the monitoring of CPU utilization and idle time.
HL-REQ-WP72-14	Core affinity	The execution environment should allow binding a function to a specific core.

Table 22: Technology Requirements

Requirement ID	**Title**	**Description**
HL-TREQ-WP72-01	Mixed-criticality run time environments	The platform shall support two independent run time environments with distinct levels of criticality.
HL-TREQ-WP72-02	Mixed-criticality hardware support	The hardware platform shall support hardware virtualization.
HL-TREQ-WP72-03	Low criticality real time task CPU isolation	The critical execution environment shall support task isolation for low criticality real time tasks by using the CPU's mode USER while the kernel uses the SUPERVISOR mode.
HL-TREQ-WP72-04	Low criticality real time task memory isolation	The execution environment shall use virtual addressing for low criticality tasks.
HL-TREQ-WP72-05	Low criticality task low level access	The low criticality tasks shall use system calls to the execution environment to access kernel's resources.
HL-TREQ-WP72-06	Task development and portability	The execution environment shall provide a C library that could be used by real time tasks to access system calls.
HL-TREQ-WP72-07	High criticality hard real time tasks	The execution environment shall support hard real time critical tasks within the same physical space as the kernel.
HL-TREQ-WP72-08	High criticality task low level access	Hard real time tasks shall be similar to kernel threads and have access to all assigned hardware and kernel resources.
HL-TREQ-WP72-09	Concurrent runtime environments	The platform shall support the concurrent execution of two independent run time environments.
HL-TREQ-WP72-10	Inter-RTE communication	The software containment component shall support two-way data transfer between both critical and non-critical RTE.
HL-TREQ-WP72-11	Inter-RTE communication isolation	The software containment shall support the CPUs MONITOR mode to securely switch between RTE.
HL-TREQ-WP72-12	Low criticality RTE access	The software containment shall support hyper/monitor calls to allow non-critical RTE access the secure kernel resources.
HL-TREQ-WP72-13	Critical RTE access to non-critical core	The software containment shall support a method of forcing a target non-critical core to switch to the critical RTE.
HL-TREQ-WP72-14	Hard real time scheduling	The RTOS shall support the scheduling of hard real time critical tasks.

HL-TREQ-WP72-15	Real time less-critical task scheduling	The RTOS shall support the scheduling of real time less-critical tasks.
HL-TREQ-WP72-16	RTOS system calls	The RTOS shall support system calls for less-critical tasks and direct calls to hard real time critical tasks.
HL-TREQ-WP72-17	RTOS memory management	The RTOS shall support memory management for both physically addressing critical hard real time tasks and virtual addressing less-critical real time tasks.
HL-TREQ-WP72-18	RTOS interrupt manager	The RTOS shall support managing both critical and non-critical interrupts sources and targets as well their priority and intended RTE.
HL-TREQ-WP72-19	RTOS C library	The RTOS shall support a C library responsible for accessing the kernel resources through system calls.
HL-TREQ-WP72-20	Device Manager	Each device shall be previously associated and the hardware platform configured on boot by the critical RTOS.
HL-TREQ-WP72-21	Hypervisor isolation	The hypervisor shall support isolation from non-critical RTE by using the CPU’s MONITOR mode.
HL-TREQ-WP72-22	Hypervisor service access	The hypervisor shall support access to its services through monitor calls.
HL-TREQ-WP72-23	Hypervisor API	The hypervisor component shall deliver a C library for a portable access to hypervisor services.
HL-TREQ-WP72-24	Hypervisor resource management	The hypervisor shall support at runtime hardware platform security configuration.
HL-TREQ-WP72-25	Hypervisor non-critical RTE boot	The hypervisor shall support booting the non-critical Android RTE.
HL-TREQ-WP72-26	Hypervisor non-critical RTE stop	The hypervisor shall support stopping the non-critical Android RTE at runtime.
HL-TREQ-WP72-27	Hypervisor non-critical RTE monitoring	The hypervisor shall support non-critical RTE status monitoring.
HL-TREQ-WP72-28	Critical RTE hardware access	The safety critical RTE shall have full access to the hardware platform.
HL-TREQ-WP72-29	Hardware devices security	Each device shall have its criticality pre-determined as well the possibility of its reassignment.
HL-TREQ-WP72-30	Non-critical hardware access	The non-critical RTE shall have direct access to devices associated with it.
HL-TREQ-WP72-31	Critical hardware share	The safety critical RTE shall support hardware sharing through a secure communication.
HL-TREQ-WP72-32	Resource reassignment	The platform shall support the reassignment of a non-critical hardware device to the safety critical RTE.
HL-TREQ-WP72-33	Inter-core communication	The execution environment shall support an inter-core communication mechanism.
HL-TREQ-WP72-34	Multiple target cores	The inter-core component shall support single/multiple target core(s).
HL-TREQ-WP72-35	Secure communication	The inter-core component shall support secure notifications.
HL-TREQ-WP72-36	Inter-core communication access	The inter-core component shall support access from/to both critical and non-critical RTE.
HL-TREQ-WP72-37	Inter-core communication	The inter-core component shall support isolation of internal components from non-critical software.

	isolation	
HL-TREQ-WP72-38	Platform monitoring	The safety critical RTE shall support a monitoring platform that each component can access to store its data.
HL-TREQ-WP72-39	Platform fault injection	The safety critical RTE shall support a fault injection platform that each component can register its faults points.
HL-TREQ-WP72-40	Scheduler monitoring	The RTOS scheduler shall support monitoring safety critical core usage and deadline failures.
HL-TREQ-WP72-41	Memory management monitoring	The RTOS memory manager shall support monitoring kernel and task memory usage as well memory faults.
HL-TREQ-WP72-42	Interrupt management monitoring	The RTOS interrupt manager shall support monitoring the number of interrupts received and time consumed.
HL-TREQ-WP72-43	Inter-core communication monitoring	The inter-core communication shall support monitoring the number of open channels.
HL-TREQ-WP72-44	Infotainment monitoring	The infotainment RTE shall support monitoring of execution status.
HL-TREQ-WP72-45	Scheduler fault-injection	The scheduler shall support fault injection to test correct task preemption.
HL-TREQ-WP72-46	Memory management fault-injection	The memory management shall support fault injection to test for out-of-memory and memory access protection conditions.
HL-TREQ-WP72-47	Interrupt management fault-injection	The interrupt manager shall support fault injection to test for correct interrupt/preemption isolation.
HL-TREQ-WP72-48	RTE core affinity	The platform shall support the attribution of a number of cores to a specific RTE.

9.2.2 ST7.5.2 Safety Aspects of Multi-criticality Applications

This section describes the safety activities performed to ensure the demonstrator compliance with the ISO26262 standard.

Safety management during product development includes the planning and coordination of the safety activities, the progression of the safety lifecycle, the creation of the safety case, and the execution of the confirmation measures. In accordance with the ISO26262 standard, the safety plan specifies the planning of the safety activities for the development of automotive products. However, ensuring full compliance with the activities required by a complete safety life-cycle is not feasible within the context of the current project. Instead, we have tailored the safety related activities to focus on two key activities to support the architectural design definition: hazard analysis and risk assessment, and safety analysis.

9.2.2.1 Hazard Analysis and Risk Assessment

A hazard analysis and risk assessment was performed to identify and to categorise the hazards that malfunctions in the demonstrator components can trigger and to formulate the safety goals related to the prevention or mitigation of the hazardous events, in order to avoid unreasonable risk. The Hazard Analysis and Risk Assessment started at early stages of the use case, during the conception phase of the demonstrator architectural design as part of sub-task ST7.5.2.

The results of the safety analyses indicate if the respective safety goals or safety requirements were complied with or not. If a safety goal or a safety requirement is not complied with, the results of the safety analysis were used to derive prevention, or detection, or effect mitigation measures regarding the faults or failures causing the violation. Several measures deriving from the safety analyses were implemented as part of the demonstrator software development. These results were also useful for deriving new test cases and for improving existing ones.

The results of the Hazard Analysis and Risk Assessment as well as the identification of the Safety Goals for the safety-critical RTOS are going to be detailed in the platform's Hazard Analysis and Risk Assessment Report.

9.2.2.2 Safety Analysis (FMEA)

A safety analysis was performed, through the application of the FMEA technique, to examine the consequences of the faults and failures of the functions, behaviour and design of items and elements. This analysis also provides information on conditions and causes that could lead to the violation of a safety goal or safety requirement.

Additionally, the safety analysis can also contribute to the identification of new hazards not previously identified during the hazard analysis and risk assessment.

A qualitative safety analysis has supported the architectural design of the safety-critical RTOS through the application of the FMEA technique.

The safety analyses performed during the software architectural design and the results are being recorded in the Hazard Analysis and Risk Assessment Report.

9.2.3 ST7.5.2 Integration of HW and SW Platform to support Multi-Criticality

Under this task the design and implementation of a small, multi-core, mixed-criticality, RTOS runtime environment (RTE) was performed. The mixed-criticality RTOS provides a solution that enables software with distinct safety-critical requirements to run concurrently on a multi-core hardware platform. The RTOS is capable of managing safety-critical tasks that run concurrently with a commercial off-the-shelf (COTS) general purpose operating system with infotainment capabilities. This is achieved by means of a Hypervisor that manages a virtualized environment in which the COTS OS runs and shares resources safely (i.e., without interfering) with the safety-critical tasks.

The implementation of mixed-criticality applications requires strong partitioning of applications, while still maintaining good performance and compliance with the automotive standard ISO26262. To fulfil these requirements, the Freescale SABRE i.MX6 Quad System on Chip (SOC) was selected as the hardware platform, which provides a hardware virtualization solution already certified for automotive usage. This enabled a small virtualization software footprint by the usage of TrustZone®, a hardware enforced container by ARM® available on i.MX6 SOC.

Figure 64 provides a high level description of how the RTOS was integrated with the hardware platform. The architecture consists of two runtime environments (RTE) with distinct levels of criticality running concurrently on a multicore platform with each RTE having its own operating system (OS) managing the associated tasks and resources.

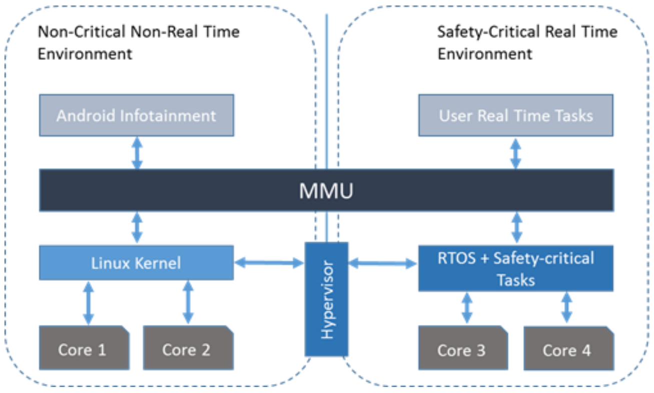

Non-Critical Non-Real Time Environment
Safety-Critical Real Time Environment
Android Infotainment
User Real Time Tasks
MMU
Linux Kernel
Hypervisor
RTOS + Safety-critical Tasks
Core 1
Core 2
Core 3
Core 4

Figure 64: Run Time Environment for Multi-Core and Multi-Criticality.

In the real time environment, a safety-critical RTOS will handle the scheduling of the safety-critical real time tasks and user realtime tasks. Its kernel supports the necessary safety features to guarantee that the user realtime tasks do not interfere with any RTOS components under all circumstances, even in the case of spurious events such as a crash. This isolation is achieved and enforced by the following mechanisms:

- Memory Management Unit (MMU) for access protection that allows space partitioning within a virtualized memory address (VMA) space;
- CPU hierarchical user/supervisor privileges
- Access to systems' resources only through system calls (RTOS) and hypervisor calls (Linux).

The safety-critical realtime tasks run within the same space as the kernel and use the kernel APIs through direct calls. Access to resources however, is also made through system calls in order to correctly handle resource sharing and subsequent task wait conditions.

The RTOS device management secures the necessary system's resources for both RTE. The guest OS through the hypervisor then requests access to them through a secure communication layer between them.

In the non-real time environment, Android will be the OS responsible for the infotainment platform and associated graphical user interface (GUI). It will be contained using hardware enforced partitioning with all non-critical resources managed by its Linux kernel. Access is limited to devices configured at boot time by the safety-critical RTE as defined by the system's designer.

Figure 64 depicts a possible system CPU cores partitioning. Other similar arrangements may be possible in the final architecture by changing the number of dedicated cores to each RTE.

9.2.3.1 Hardware technology

The domonstrator's hardware technology relies on the ARM® TrustZone® technology, which provides mechanisms that allow the required time and space partitioning necessary for a mixed-criticality platform to be developed upon. This technology enables hardware enforced isolation through the creation of an additional level of partitioning above all the existing ones, allowing the creation of two zones or, in its nomenclature, a secure and normal "world". A brief description of those features is provided in the following subsections.

One of these technologies is shown in Figure 65, with all CPU modes split and a new CPU mode (monitor) responsible for the secure switching between execution modes or "worlds". This adds not only a new layer of separation between user code and kernel code but does so in two distinct modes enabling the execution of two distinct operating systems with minimal changes. One executing in a privileged CPU mode („secure“) and the other on an unprivileged mode („normal“) both supporting their own exception handlers allowing the correct OS to handle them internally.

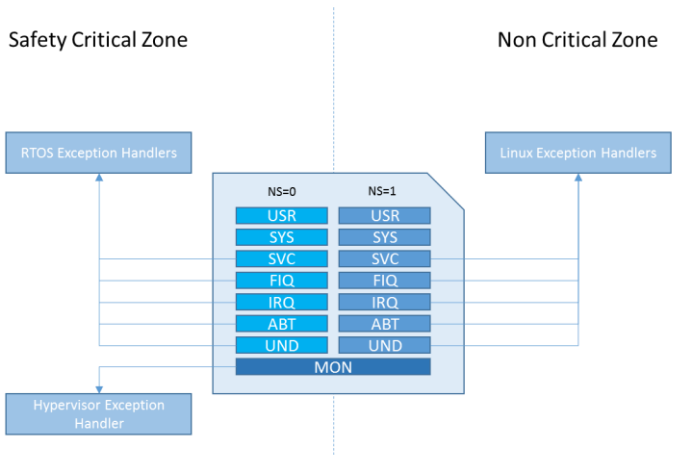


Figure 65: ARM TrustZone CPU modes

The hardware architecture allows to control the access to the system resources. This is also an important feature for mixed-criticality systems, because it allows to restrict the access of non-critical application to critical system resources other than the CPU, such as hardware devices. The high level hardware architecture is shown in Figure 66. Depending on the security configuration, it is possible to restrict the access to the hardware resources through either the address space controller or the security unit, based on the source of the request. More specifically, the following hardware components and features allow to enforce the required time and space partitioning:

- **Bus and Device Aware Mode**: Supporting buses and devices are aware of current executing world and allow access to be restricted according to design requirements (e.g., deny access to non-critical RTE, but allow it to the safety-critical RTE);
- **Address Space Controller**: Memory access monitoring hardware on top of the MMU ensures memory regions protection with configurable zones for R/W access control;
- **Interrupt Controller**: Secure interrupts allows the safety-critical RTE to take back control of a CPU core at any time from non-critical RTE control even in case it may be unresponsive/crashed. They're also handled solely by the safety-critical RTE;

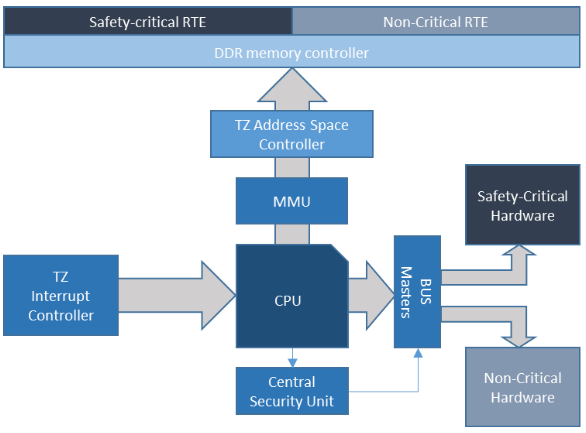


Figure 66: Used Trustzone Hardware Components

9.2.3.2 Software Architecture

Figure 67 provides an overview of all the demonstrator's main architectural components and sub-components. Those components are briefly described in the following subsections.

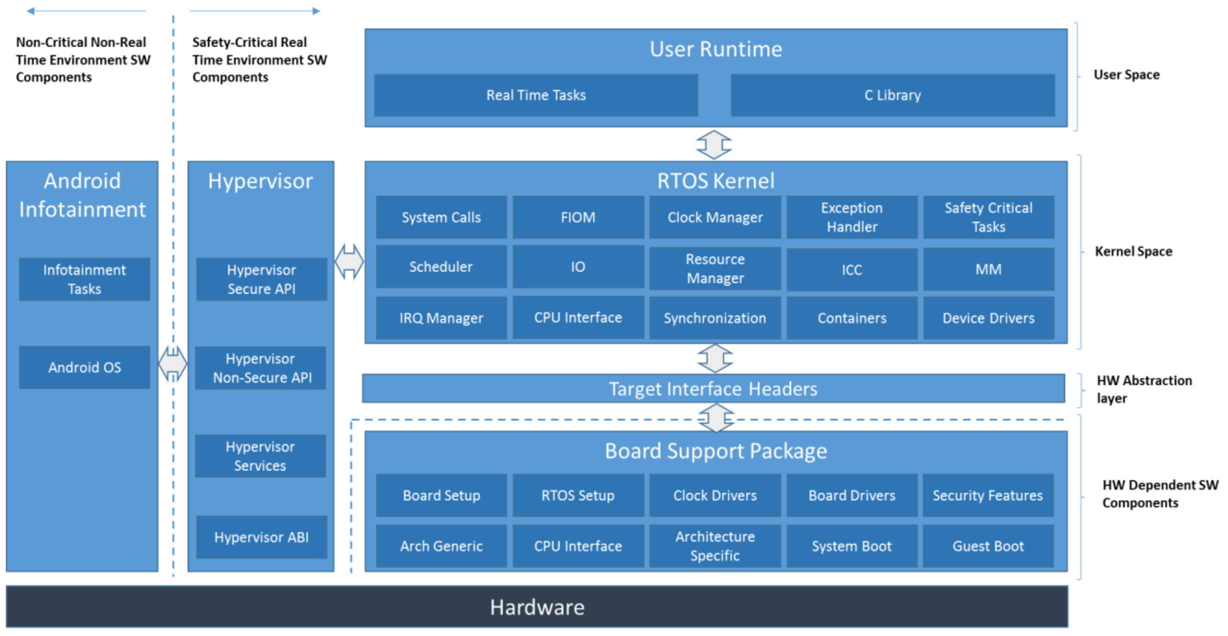


Figure 67: Demonstrator Architecture

9.2.3.2.1 Android Infotainment

A general purpose operating system running non-critical non-real time applications can run safely without interfering with the safety-critical real time environment. The ANDROID OS as a feature rich environment with great user interaction is the base platform for the infotainment system. Using the software containment mechanisms implemented by the “Hypervisor” and supported by the hardware enforced system partitioning technology, the guest OS runs within a hardware enforced software container in an unprivileged, guest mode.

In order to support correct execution under the mentioned technologies, the Linux kernel is modified to accommodate the necessary virtualization requirements such as virtual device drivers as well the secure communication mechanism using the Hypervisor API.

9.2.3.2.2 Hypervisor

Responsible to configure the hardware that provides the isolation between the two environments. To interact and control the guest OS, both the RTOS kernel and safety-critical tasks access the hypervisor features directly by an API (Hypervisor Secure API). This API acts directly over the Hypervisor Services to perform actions such as IO configuration, guest OS start, guest OS stop, guest OS pause, etc. The Hypervisor Service is the core of the hypervisor component being responsible for configuring guest OS parameters (MMU setup), boot, handling of hypervisor calls and their interactions with the RTOS Kernel, and providing console output to the guest OS. The Hypervisor Non-Secure API is a C library that exposes the hypervisor services to the non-critical RTE and it’s used by the Linux kernel to interact with the safety-critical RTE. For each functionality, this component calls the required Hypervisor Service routine through hypercalls responsible of executing the requested action. In order to be as portable as possible all direct hardware interactions are performed through an architecture independent infrastructure, much like the BSP for the RTOS. The Hypervisor ABI is a simple C to ASM translator that calls the correct CPU instruction to trigger the required hypervisor exception call.

9.2.3.2.3 User Runtime

User real time tasks differ from safety-critical tasks only by the isolation and priority. Using the MMU and CPU user and supervisor modes, each user real time task executed under the safety-critical RTOS will run under VMA, as well on the unprivileged CPU mode (USR). This ensures correct space partitioning from safety-critical components.

The system designer must always assign a lower priority than their safety-critical counterparts, otherwise they'll would have higher priority accessing system resources.
The safety-critical RTE user applications that run in an isolated virtual memory space benefit from the features provided by the C library. Using the RTOS kernel system calls, the C library exposes several services to the user tasks.

9.2.3.2.4 RTOS Kernel

The RTOS kernel represents the core of the operating system and is the main responsible for configuring and managing all resources as well handle all task management and scheduling both safety-critical real time tasks and user real time tasks. It's composed of several layers, sub-systems and interfaces. The main components are described below:

- **Clock Management**: in a realtime environment, precise knowledge and control of time is of the outermost importance. The RTOS scheduler requires fine grained control over a clock source in order to correctly handle time as a resource and handle its partitioning between demanding tasks;
- **Interrupt Management**: a RTOS system is expected to react to external events within a predictable deadline. All these external events are translated via the hardware that collects each hardware interrupt line and connects with the CPU interrupt interface. The interrupt manager is thus responsible for:
  - Managing a list of Interrupt Service Routines (ISR) for each available interrupt source;
  - Providing an API for registering and configuring each ISR;
  - Call the corresponding ISR for each interrupt triggered to the CPU;
  - Providing an API for filtering IRQ below a certain priority (with enabled HW);
  - Clearing the interrupt enable bit after each called ISR.
- **Resource Management**: in order to correctly define and manage access to system resources, each resource needs to be attached to a permission protocol and subsequent management and access API. Sharing of such resources demands a sharing protocol to be defined, one which guarantees predictable worst-case access in mixed-criticality systems. Therefore, an adaptation of the MC-IPC[2] protocol was implemented to provide the required time partitioning between resource accesses by mixed-criticality tasks running on the multi-core platform.
- **Device Management**: for correctly performing the management of the hardware devices through the aforementioned resource sharing protocol, the RTOS kernel must provide the necessary interfaces for configuring, obtaining and releasing all managed devices. This functionality is especially important when considering the capability of the non-critical RTE to perform direct hardware access through the hardware virtualization mechanisms. Therefore, the RTOS kernel can be configured at design time to define the criticality level and share type of each hardware device, thus ensuring at all time that the target device is only access by the RTE with compatible criticality level.
- **Memory Management**: by using hardware enforced system partitioning, the system's memory is at first partitioned between the two RTE, where each of them is responsible for managing the availability and kernel/task protection. On the safety-critical RTE, the memory partition is further divided into static objects and pages of a predefined size. Since RTOS kernel tasks won't use dynamic memory allocations their size are fixed and set at compile time. User real-time tasks on the other hand may require additional memory. Each user real time task managed by the safety-critical RTOS will be assigned to one or more memory pages from the safety-critical partition, but isolated by the use of MMU and VMA. Internally, the task may request additional memory up to the page limit. It may request additional pages to the kernel through the C library, which will then trigger a system call. In case of memory unavailability, the request is rejected.
- **Inter-core Communication (ICC)**: the secure inter-core communication mechanism acts as a lower level communication layer that enables endpoints of distinct levels of safety-critical requirements to communicate with each other, taking into account the system configuration, target permissions and availability. It uses the hardware security and hypervisor extensions to

[2] Brandenburg, Björn B. "A synchronous IPC protocol for predictable access to shared resources in mixed-criticality systems." *Real-Time Systems Symposium (RTSS), 2014 IEEE*. IEEE, 2014.

ensure correct time and space partitioning and prevent error propagation from both user applications as well from the guest OS. The architecture consists of a series of virtual channels through which messages are delivered to their intended targets, taking into account both the requesting and target criticality levels and constrains. The channels define the target core(s) to where the request shall be added into each priority run queue.

- **Real-time Scheduler**: in order to achieve tasks with different temporal criticalities to coexist in the same platform, server-based or bandwidth reservation techniques are used to effectively ensure the temporal isolation and protection of the critical tasks, while still respecting all realtime constraints of the applications. Servers allow for the reservation of a portion of the embedded system capacity for a specific application. Therefore it allows applications to run independently through time partitioning. The servers are fully preemtable and are assigned a fixed-priority. Within each server (or time partition), the tasks can be scheduled with a scheduling policy that can be different from the top-level scheduler scheduling policy.
- **System Calls**: the RTOS provides access to system's resources through system calls to the kernel. They provide the required space partitioning by isolating system's internals into a different memory space (non-VMA) and CPU mode of execution. System calls provide functionality to several of the RTOS sub-systems. They allow tasks to perform IO onto devices, hardware operations such as system reboot or power off and task management operations such as sleep and execution yield.

9.2.3.2.5 Boad Support Package (BSP) abstraction layer

All interactions with hardware functionalities use abstract functions provided by a hardware abstraction layer called Target Interface Headers that ensures portability to other hardware architectures. This abstraction layer allows an easier debugging, maintainability and portability. The target interface headers declare various functions the RTOS Kernel expect the BSP to implement.

9.2.3.2.6 Boad Support Package (BSP)

The BSP is the hardware dependent layer responsible for basic functionality such as: hardware access, boot process, RTOS setup, context switching, task synchronization primitives, etc. The main functionalities implemented in the BSP are:

- Architecture specific functions;
- Atomic operations;
- Bit operations;
- Handle task context switching;
- Hardware RTC;
- Interrupt Request (IRQ) Management;
- Memory operations;
- Exception handling;
- Synchronization primitives;
- Device interfaces;
- Onboard device drivers and device tree population.

9.2.4 ST7.5.2 Integration between Infotainment and eCall Applications

Due to changes in the consortium compositions, instead of an eCall application, a crash detection and notification user task was implemented on top of the RTOS userspace.
This task provides one of the use case's goals of sending an emergency notification in the event of the vehicle crash. The integration of resources to functionalities enables the use case demonstrator to showcase resource and information sharing between two applications of distinct levels of criticality.
In order to showcase the two tier task time and space partitioning the process is divided into two main steps:

- A safety-critical task constantly monitors the onboard acceleration sensor and in case of impact, a high acceleration is detected and the crash warning system is triggered:

- The infotainment system is stopped through a call to the hypervisor services by the safety-critical task which, using ICC, will redirect the call to the hypervisor instance in the target CPU core and carry out the guest OS stoppage.
- All hardware devices are immediately secured using the provided hardware security extensions and their permissions updated accordingly.

- The user real time application is signalled through IPC (by the safety-critical task) that a crash event has occurred and a notification is sent. Using the C library it opens the notification device and sends out all information to the designed output.

9.2.5 ST7.5.2 Use Case Demonstrator

In this section we provide an overview of some of the demonstrator showcase technologies.

Figure 68 depicts the platform boot sequence with all matching high level requirements implemented on each boot process step.

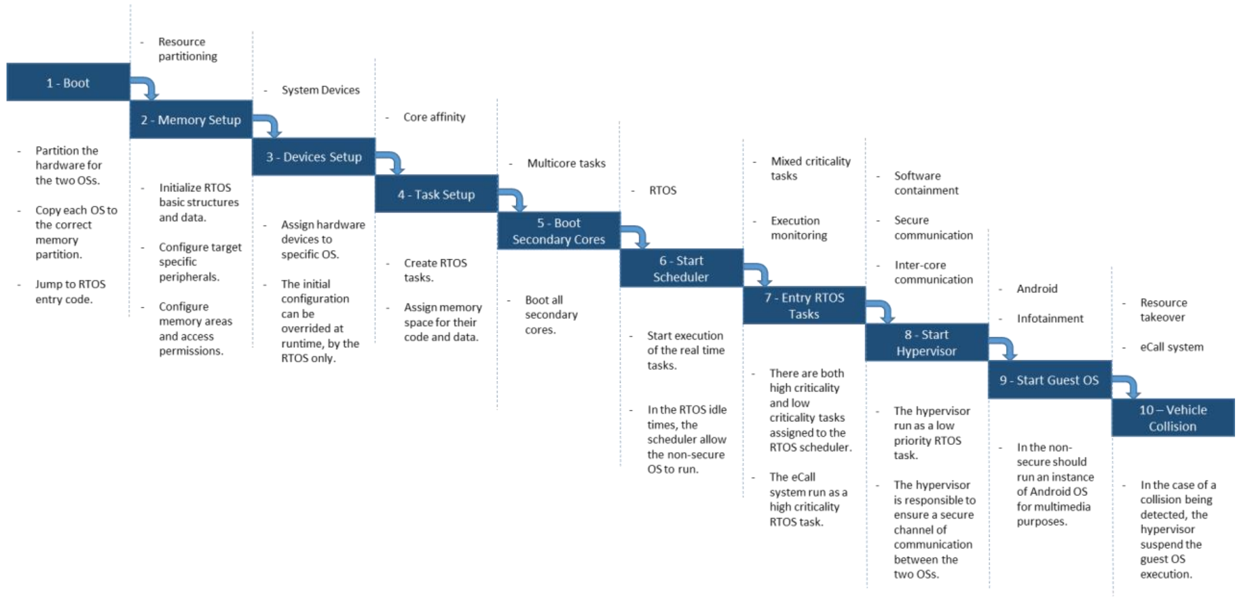

1 - Boot
- Partition the hardware for the two OSs.
- Copy each OS to the correct memory partition.
- Jump to RTOS entry code.
- Resource partitioning
2 - Memory Setup
- Initialize RTOS basic structures and data.
- Configure target specific peripherals.
- Configure memory areas and access permissions.
- System Devices
3 - Devices Setup
- Assign hardware devices to specific OS.
- The initial configuration can be overrided at runtime, by the RTOS only.
- Core affinity
4 - Task Setup
- Create RTOS tasks.
- Assign memory space for their code and data.
- Multicore tasks
5 - Boot Secondary Cores
- Boot all secondary cores.
- RTOS
6 - Start Scheduler
- Start execution of the real time tasks.
- In the RTOS idle times, the scheduler allow the non-secure OS to run.
- Mixed criticality tasks
- Execution monitoring
7 - Entry RTOS Tasks
- There are both high criticality and low criticality tasks assigned to the RTOS scheduler.
- The eCall system run as a high criticality RTOS task.
- Software containment
- Secure communication
- Inter-core communication
8 - Start Hypervisor
- The hypervisor run as a low priority RTOS task.
- The hypervisor is responsible to ensure a secure channel of communication between the two OSs.
- Android
- Infotainment
9 - Start Guest OS
- In the non-secure should run an instance of Android OS for multimedia purposes.
- Resource takeover
- eCall system
10 – Vehicle Collision
- In the case of a collision being detected, the hypervisor suspend the guest OS execution.

Figure 68: Use Case Demonstrator Main Sequence

The system boots a single CPU core and starts partitioning the memory and copy each OS to the correct region. After which, the bootloader jump to the RTOS entry point, giving it control over the boot CPU core. The RTOS then proceeds to initialize its modules and internal structures necessary to support the setup process by the target specific initialization code. As an embedded solution the RTOS objects and settings for each interface, bus, device and task are all tailored to that product deployment, as such, it’s the board support package responsibility to enumerate all supported devices, all user and kernel tasks and access policies to each resource.
When the setup process is done, the RTOS boots the remaining CPU cores and waits for each target specific initialization.
With all the configurations and initializations made, the real time scheduler is started on each CPU core configured for RTOS task scheduling. Tasks are run on their configured CPU core with their designed priority and criticality group. Each user task runs on its own virtual address space while system tasks run on physical address space.
Two of those tasks are the hypervisor main and secondary tasks. The first is responsible of setting up the guest runtime environment (RTE), while the second is used to encapsulate the secondary CPU core usage by the guest RTE. Both are used to allow the RTOs scheduler control over the amount of time the guest RTE has on each of the two CPU cores and to allow the containment of any fault coming from the hypervisor to be restricted to itself.

Once the guest RTE is properly configured with all devices designed to be accessed by the guest OS configured with access policies that enables it to do so, the hypervisor passes control over to the guest OS kernel's entry point so it can start its own boot process on the paravirtualized environment.

To present in more detail some of the technologies implemented in WP3 and integrated into T7.5, a few more demonstrations were created.

One of the technologies implemented was the Fault Injection Online Monitoring. Its key features are:

- Hability to inject faults in sections otherwise would require hardware faults
- Provides a monitoring API for tasks to monitor critical sections of the system

To demonstrate the fault injection feature a user task was created that requests the kernel for more memory though the page allocation system call. The FIOM is configured to trigger only on the second iteration on which it will return an invalid value. Hence, on the first page allocation no fault is triggered when the task accesses the new allocated page but on the second the MMU triggers a memory access exception resulting in the faulty task stoppage.

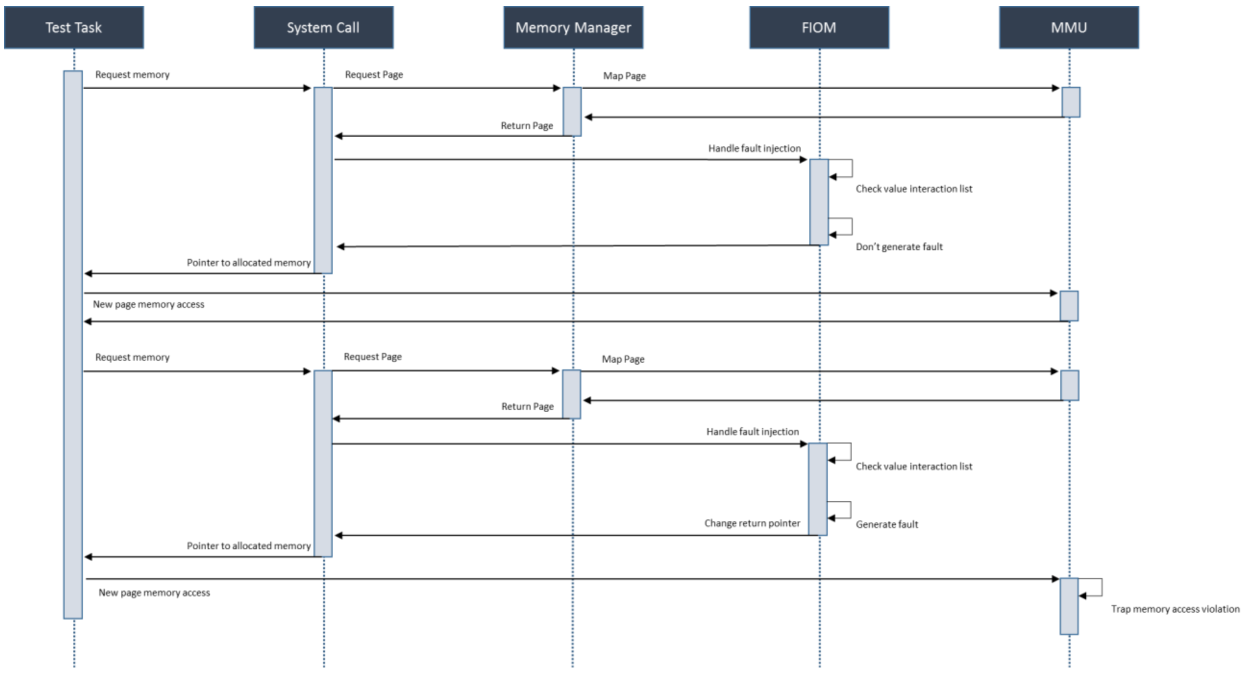


Figure 69: FIOM Demonstrator Sequence

Next, to demonstrate the Inter-Core Communication mechanism the RTOS command shell is used to send remote procedure call to another CPU core to halt a specific task as shown in Figure 70.

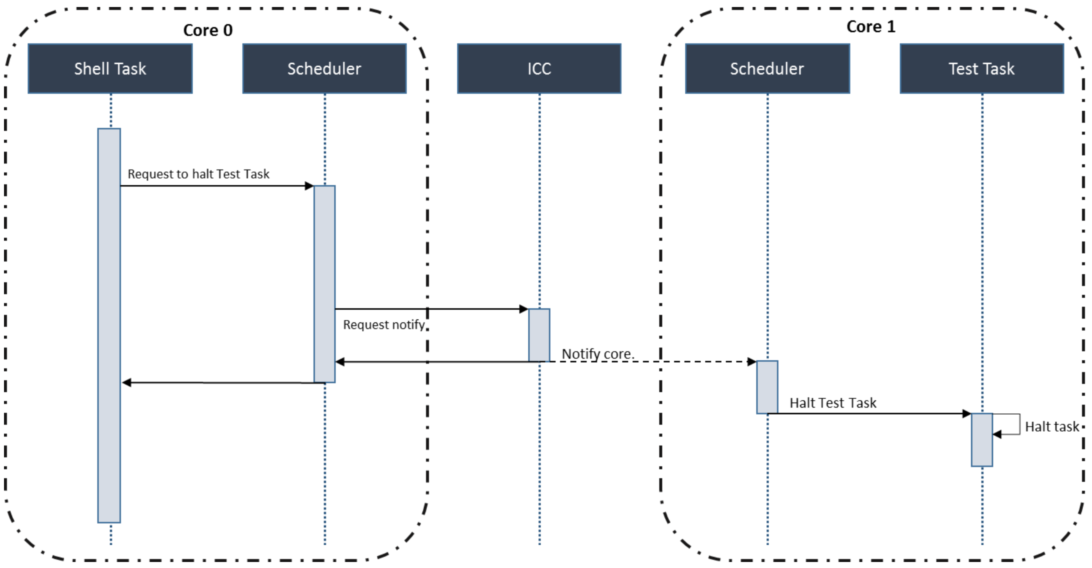


Figure 70: ICC Demonstrator Sequence

The remote procedure call is required because the platform RTOS scheduler is a per-core local scheduler. As such to interfere with the scheduling on a remote processor the platform uses the ICC to send remote procedure calls to signal the target scheduler to perform the desired action. In the case of this demonstrator the RTOS command shell task is used to send a task halt request to a task running on a different CPU core than the one used by the shell task. The status can be then shown using again the command shell task to list all tasks and their respective status.

9.3 Exploitation

The resulting platform developed in T7.5 shall be evaluated as a mixed-criticality infotainment product, capable of running simultaneously a non-critical user interface with user comfort features, as well both high and low safety-critical, real time functionalities on a multi-core platform. This evaluation will factor the final platform certifiability according to customer's quality and safety requirements, platform features provided and finaly its performance.

The initial expectation of market introduction in late 2016 was re-evaluated, and now a new estimate to late 2017 is projected, mainly due to the integration phase requiring more V&V effort than originally expected.

Regarding the market potential, it is expected a volume in the order of hundreds of development licenses per year, with a market share of 0,5% up to 2% for the multi-core mixed-criticality platforms, again vastly depending on the TRL milestones.

10. Appendix 6: T7.6 UC Next Generation Electronic Architecture for Commercial Vehicles

10.1 Relation between Business Needs and KPIs

The business needs behind this use case are described in the table below:

Table 23: Business needs for use case "Next generation electronic architecture for commercial vehicles"

No	Title	Description
BN_T7.6_01	Number of control units	Reduction of the total number of ECUs and the variants across the ECUs with respect to the current generation E/E architecture to reduce cost and complexity as well as to increase performance.
BN_T7.6_02	Architecture evaluation methods	Investigation of Systematic evaluation methods to compare alternative E/E architectures, including embedded network communications, to identify the most suitable one that meets a given set of requirements as well as supports scalability, flexibility, reusability and adaptability.
BN_T7.6_03	Partitioning and allocation	Exploration of concepts, methods and tools for efficient partitioning and allocation of functionalities as well as software across the multicore ECUs at the E/E architecture level and across the processor cores within a particular ECU to maximize performance and resource utilization.
BN_T7.6_04	Dependability enhancement	Enhancement of dependability (functional safety, reliability, fault tolerance, security, and availability) both at the E/E architecture level and at the ECU level by utilization of available resources of multicores in order to improve product quality and safety.
BN_T7.6_05	Mixed criticality support	Ensure separation, i.e. freedom from interference among applications with different criticality levels (as defined in ISO 26262) and manage shared resources of multicore-based ECUs.
BN_T7.6_06	Communications	Investigation of concepts for Inter-ECU at the E/E architecture level and intra-ECU communication (including topics network topology, protocol, bandwidth, delay, technology) in multicore based architectures.
BN_T7.6_07	Service-oriented architecture	Viability and applicability of concepts and techniques related to Service-oriented Architecture (SoA) for real-time functional safety-related automotive systems.
BN_T7.6_08	Runtime optimization	Viability and applicability of concepts and techniques related to run-time optimizations for real-time safety-related automotive systems.
BN_T7.6_09	AUTOSAR support for multicore, SoA and runtime optimization.	Recommendations to (a) improve multicore support for mixed-critical systems in future releases of AUTOSAR as well as other relevant standards and specifications, and (b) suggest required changes in AUTOSAR to support SoA and run-time optimizations.
BN_T7.6_10	Mixed OS support	Investigation of concepts for mixed OS support in one and the same ECU.

The relation between the KPIs and the Business Needs is shown in the table below:

Table 24: KPIs – Business Needs relation for use case "Next generation electronic architecture for commercial vehicles"

No	Title	Description	BN relation
KPI_T7.6_01	Reduction of number of control units	Reduced total number of ECUs in next generation E/E architecture for commercial vehicles by 20% with the respect to the current-generation E/E architecture, assuming same level of functionalities and features realized.	BN_T7.6_01
KPI_T7.6_02	Definition of architecture evaluation methods	Defined systematic evaluation methods to compare alternative E/E architectures to identify the most suitable one that meets a given set of requirements as well as supports scalability, flexibility, reusability and adaptability. Such evaluation methods should include embedded network communications.	BN_T7.6_02
KPI_T7.6_03	Definition of concepts, methods and tools for partitioning and allocation of software	Defined concepts, methods and tools for efficient partitioning and allocation of functionalities as well as software across the multicore ECUs at the E/E architecture level and across the processor cores within a particular ECU to maximize performance and resource utilization.	BN_T7.6_03
KPI_T7.6_04	Definition of methodologies for dependability enhancement	Defined methodologies and guidelines for utilization of available resources of multicores to improve product quality and safety by enhancing dependability (functional safety, reliability, fault tolerance, security, and availability) both at the E/E architecture level and at the ECU level.	BN_T7.6_04
KPI_T7.6_05	Increase of the degree of integration of applications of mixed criticality levels	Increase integration of functionalities with different criticality levels within a single ECU by 20% with respect to the current level of functionality integration by exploiting the potential of multicores.	BN_T7.6_05
KPI_T7.6_06	Definition of communications strategies	Defined strategies for Inter-ECU (at the E/E architecture level) and intra-ECU communication (network topology, protocol, bandwidth, delay, technology).	BN_T7.6_06
KPI_T7.6_07	Definition of Service-oriented architecture concepts	Defined concepts including viability and applicability of Service-oriented Architecture (SoA) for real-time functional safety-related automotive systems.	BN_T7.6_07
KPI_T7.6_08	Definition of runtime optimization concepts	Defined concepts including viability and applicability of run-time optimizations for real-time safety-related automotive systems.	BN_T7.6_08
KPI_T7.6_09	Definition of concepts and recommendations for AUTOSAR support for multicore, SoA and runtime optimization.	Defined concepts and recommendations to (a) improve multicore support for mixed-critical systems in future releases of AUTOSAR as well as other relevant standards and specifications, and (b) suggest required changes in AUTOSAR to support SoA and run-time optimizations.	BN_T7.6_09
KPI_T7.6_10	Definition of concepts for mixed OS support	Defined concepts for mixed OS support in one and the same ECU.	BN_T7.6_10

10.2 Sub task descriptions

10.2.1 ST7.6.1 Use case description and requirements

The activities performed within this task are

- Description of the use case tasks (see D7.1, D7.13 and D7.14)
- Identification of requirements specific to the use case based on the already identified business needs.

The High Level Requirements (HLRs) shown below where identified and handed over to the WP1 in an early phase of the project.

Table 25: High level requirements for use case "Next generation electronic architecture for commercial vehicles"

Requirement ID	Short description	Description	Rationale	BN relation	Sub-task relation
HL-REQ-WP07-023	Generic computational ECUs.	Technology for generic configurable ECUs with high computational capabilities shall be developed and documented.	Traditionally new ECUs are added when new functionality is introduced and as a consequence many different ECU variants exists. High capacity generic ECUs carrying multiple applications and possible to easily configure for different kinds of applications would be of benefit in this regard.	BN_T7.6_01	ST7.6.2, ST7.6.3
HL-REQ-WP07-024	Architecture evaluation methods.	Architecture evaluation methods shall be defined to be used for comparing alternative E/E architectures, including embedded network communications, to identify the most suitable one that meets a given set of requirements as well as supports scalability, flexibility, reusability and adaptability.	To be able to evaluate architecture alternatives systematic evaluation methods is needed in order to support the selection of alternatives.	BN_T7.6_02	ST7.6.2, ST7.6.5
HL-REQ-WP07-025	Software partitioning and allocation.	Methods and tools for partitioning and allocation of software across the multicore ECUs at the E/E architecture level and across the processor cores within a particular ECU in order to maximize performance and resource utilization shall be developed.	With multicore ECUs there is a need for methods for partitioning and allocation of base SW as well as application SW across EUCs and across cores in an ECU. Such methods should be supported by tools as an integrated part of the development environment.	BN_T7.6_03	ST7.6.3
HL-REQ-	Dependability	Concepts for	Enhancement of dependability	BN_T7.6_04	ST7.6.4

Requirement ID	Short description	Description	Rationale	BN relation	Sub-task relation
WP07-026	enhancement.	dependability (functional safety, reliability, fault tolerance, security, and availability), both at the E/E architecture level and at the ECU level, shall be defined in the context of available resources of multicores.	in the multicore context in order to ensure and improve product quality and safety.		
HL-REQ-WP07-027	Mixed criticality support.	Separation, i.e. freedom from interferences, among applications with different criticality levels (as defined in ISO 26262) shall be ensured in one and the same ECU.	The integrity of applications when executing different applications at different cores these applications must be guaranteed.	BN_T7.6_05	ST7.6.3, ST7.6.4
HL-REQ-WP07-028	Communications guidelines.	Guidelines for Inter-ECU at the E/E architecture level and intra-ECU communication (including topics network topology, protocol, bandwidth, delay, technology) in multicore based architectures shall be defined.	With new architecture concepts based on multicore ECUs, new demands on the communications system can be foreseen. E.g. highly critical application communication mixed with less critical communication may raise the needs of dynamically allocated high priority channels for event driven data on the account of low priority data.	BN_T7.6_06	ST7.6.2
HL-REQ-WP07-029	Service-oriented architecture concepts.	Concepts and techniques related to Service-oriented Architecture (SoA) for real-time functional safety-related automotive systems shall be defined.	SOA for embedded systems is a new interesting approach that needs to be investigated in the light of new architecture concepts with central computational nodes and light-weight I/O nodes in order to make implementation af applications more independent of underlaying platform.	BN_T7.6_07	ST7.6.5
HL-REQ-WP07-030	Runtime optimization.	Concepts and techniques related to run-time optimizations for real-time safety-related automotive systems shall be defined.	Multicore support with high performance capabilities gives the possibility to have faster and more precise control algorithms that can be adapted and optimized during run-time.	BN_T7.6_08	ST7.6.5
HL-REQ-WP07-031	AUTOSAR support for multicore, SoA and runtime optimization.	Recommendations shall be formulated to (a) improve multicore support for mixed-critical systems in future releases of AUTOSAR as well	Any implications of SOA and run-time optimization on the AUTOSAR standard can be presented to the AUTOSAR consortium for possible standard modifications.	BN_T7.6_09	ST7.6.2, ST7.6.3, ST7.6.4, ST7.6.5

Requirement ID	Short description	Description	Rationale	BN relation	Sub-task relation
		as other relevant standards and specifications, and (b) suggest required changes in AUTOSAR to support SoA and run-time optimizations.			
HL-REQ-WP07-032	Mixed OS support.	Mixed OS, i.e. AUTOSAR OS and RT Linux, shall be supported in one and the same ECU.	Autosar is needed in order to communicate on the vehicle bus. Autosar is however not enough to run many of the new applications with demands on different communication stacks, local databases and other functionality that is part of a complex (RT)OS.	BN_T7.6_10	ST7.6.2, ST7.6.3, ST7.6.4

10.2.2 **ST7.6.2 E/E architecture for commercial vehicles**

This subtask contains two main activities:

- Evaluation of architectures
- Architecture concepts

Evaluation of architectures

The first step of architecture evaluation is to decide the criteria to be used. There are several publications that present criteria to use, most set of criteria are quite similar, but there are differences. In this project we decided to start from a military standard, mainly because it appeared the most mature and complete. When reviewing the criteria it was concluded that some criteria were not applicable and other criteria needed to be added, mainly because of different conditions and priorities for military and commercial vehicles. Also different companies within the same domain can have slightly different priorities depending on business models, strategies, existing strength and weaknesses etc. The conclusion is that you should select the criteria that are applicable for you; there is no given answer on the exact right criteria, though a majority of criteria are likely to be the same when comparing with publications and what other companies use.

Some commonly used critera are a bit vague, very near synonyms and/or overlapping. For evaluation purposes it is preferred to clearly define criteria and avoid too much overlap. For example we prefer not to use flexibility as a criterion, since this can mean many different things and should perhaps be seen as the collection of several other criteria. Instead we use more precise terms like scalability, adaptability and upgradeability. Even for these terms it is necessary to more precisely define their meaning in order to have a common view and identify whether there is an overlap or not.

Another aspect is that different criteria are not of equal importance, so it is important to define weights for criteria. It can be difficult, and you should expect results to be approximate, but weights can still indicate that one criterion is e.g. about equal, twice or ten times as important as another. Particularly some criteria are usually found to have so low weight that it is not worth the effort to include them in the evaluation, i.e. the maximum impact of the criteria is less than expected measurement accuracy. It is recommended to not have too many criteria, but only evaluate using the subset of the really important criteria.

For evaluation purpose it is very much preferred to define objective measurements rather than relying on subjective judgment. The typical objective measure would be cost. There are however two problems with defining objective measures. Firstly you typically want to perform evaluations in an early phase, in which case it can be very difficult to get good cost estimations. Commercial vehicles is perhaps particular tricky, because sometimes dedicated components/technology needs to be developed, the installation process is quite complex, and it shall still be mass produced. From our case study, it was even very difficult to collect sufficient data even for existing architectures. To compare exiting architecture with conceptual architectures is even more difficult. The second problem is that many criteria, like adaptability and scalability, are quite difficult to transform to an objective measure. This means that when comparing two architecture concepts, you can perhaps tell whether one or the other is better for each criterion, but it is difficult to quantify the impact of the differences. It is possible to evaluate subjectively, but you may or may not trust results, it is very easy to unconciuously be biased or to sub-optimise when evaluating subjectively.

Apart from the evaluation itself being difficult, there is also an effort issue with working with several architecture concepts in parallel, particularly if the intention is to reach a level where accurate evaluations can be performed. The pragmatic approach we ended up with was to use the evaluation criteria primarily as driving criteria, i.e. what we strive for when defining architecture concepts. In an early sketch phase, we can still have several architecture alternatives, but instead of performing an evaluation based on measurements, the criteria become the tool for thinking, reasoning and conceptualisation of the architecture alternatives. In our case there was one main architecture concept already after this early reasoning, but there are many sub-areas where different concepts are to be evaluated as the work progress. One could also continue to work with several complete architecture concepts in parallel. As design and prototyping progress, it becomes possible to get more data, perform measurements and more objectively evaluate according to the criteria in order to confirm whether the early reasonings were valid or not.

Architecture concepts

Current electronic architectures in vehicles typically have quite many ECUs, with multiple communication networks, and complex software distribution. There are several drawbacks of this approach, particulary the complexity of the architecture is an obstacle that slows down development of new and improved functions; it is difficult to efficiently utilize computational resources; it is costly to maintain the many number of unique hardware parts; and it is difficult to fit all electronics in the vehicle.

The Figure 71 shows a disguised example of a truck architecture of today. It uses CAN as the main communications standard, but also other standards are used, and the ECUs are based on single core processors. Each ECU realizes mostly just one application and when adding a new application yet a new ECU is added.

For a future architecture, we try out a much simpler and cleaner architecture approach. In this approach we utilize on central computational resource, which could be one ECU or a cluster of a few ECUs. In this computational resource we put most application software. Then there is a network of I/O nodes, where I/O is connected. These I/O nodes will be very simple and generic, with a generic application protocol for communicating the I/O with the applications. The network will be a combination of Ethernet and CAN, where computational resource and demanding I/O are connected on Ethernet, whereas less demanding I/O could still be connected on CAN, with a gateway between Ethernet and CAN. One of these CAN buses could be a legacy CAN bus, where current ECUs can be connected, to allow for a gradual transformation to the new architecture.

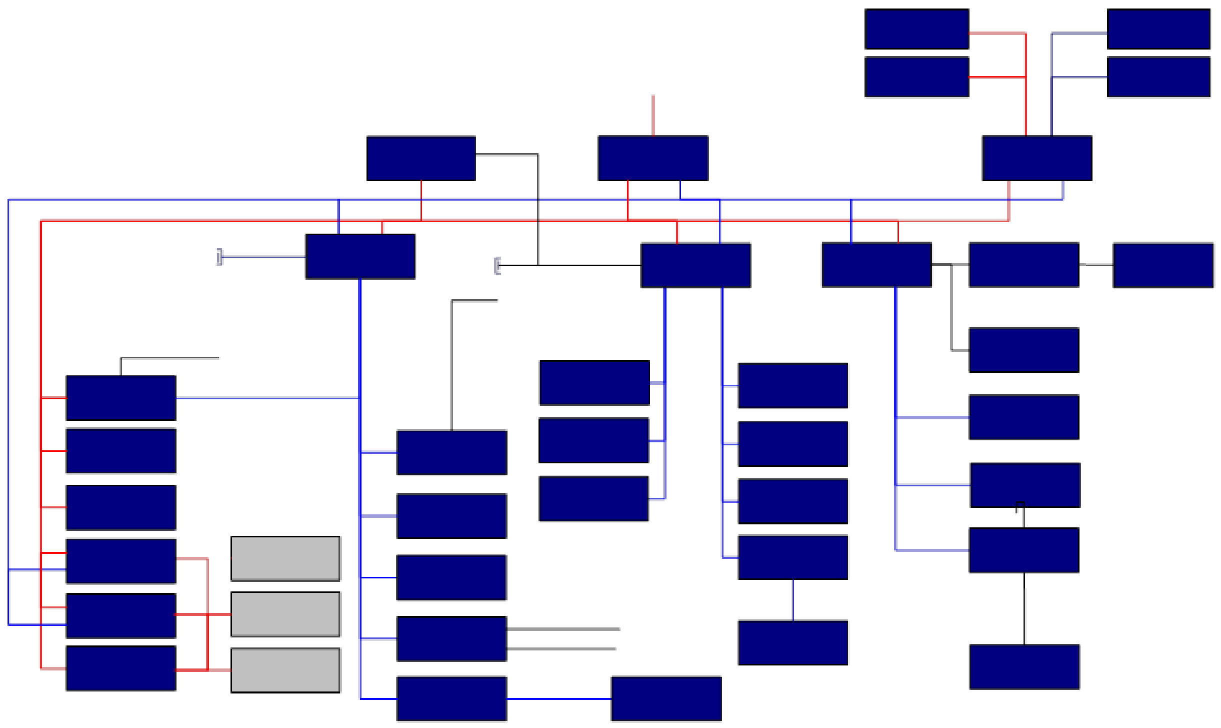


Figure 71: Example of a distributed truck E/E architecture

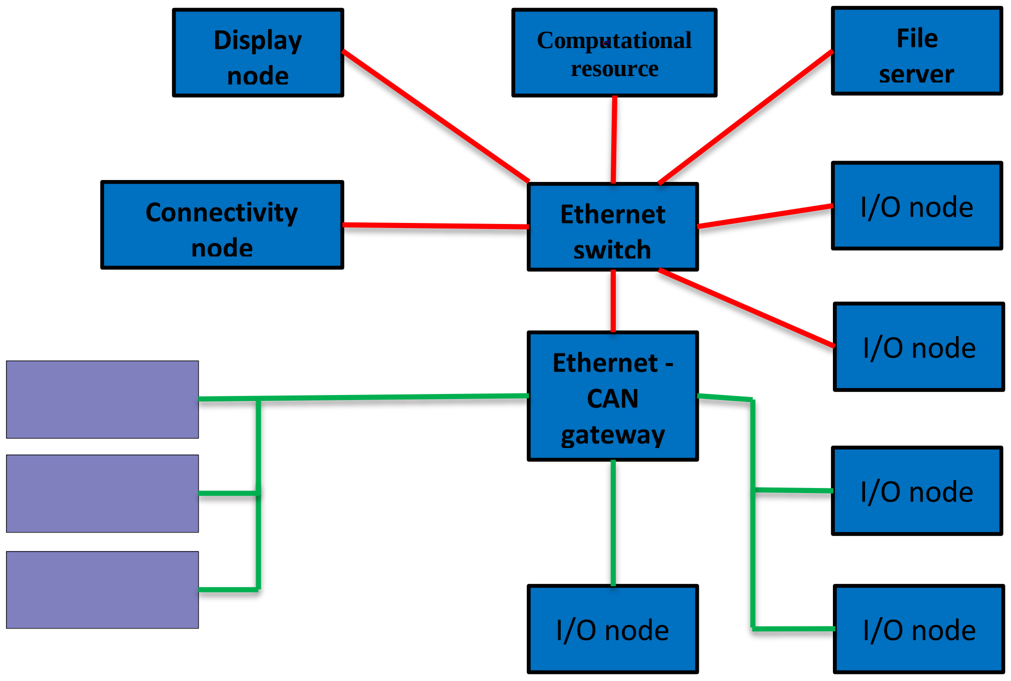


Figure 72: Principle sketch of a centralized truck E/E architecture

When building the first prototype, we are starting with as much COTS components as possible. This currently means looking at non-automotive components, since some of these technologies are not readily available in automotive components. When the concept has been demonstrated and verified, further design work is needed do develop components that are adapted for use in automotive environments.

When software is centralized, the complexity of distributing software can be minimized. New features can easily be added on existing nodes, no longer any need to add additional ECUs for new functions. The computational node can easily be scaled to required performance demands. I/O nodes with different number of I/O can be used depending on application needs, and if I/O nodes are genereic they can be reused for different purposes. The communication network can be upgraded with little impact on applications, e.g. I/O can be moved from CAN to Ethernet or Ethernet can be upgraded to higher bandwidth. This means that scalability, adaptability, upgradeability and reuasability are greatly improved.

Some challenges need to be addressed, particularly putting a lot of software in one ECU means that there is a greater need for ensuring that there is no interference between different applications, so more protection or separation mechanisms are needed. The problem of allocation is greatly reduced, but there is still a challenge to have a transparency with respect to where in the computational cluster individual application software components are executed.

Since we want to improve the software architecture, to have less complex dependencies and interfaces, there is also much work needed to transform software to fit with new architecture principles. The same goes for middleware, where we need both to adapt some existing middleware and to use or develop completely new middleware to go with the new technologies and architecture principles.

A further step is to define how architecture can be adapted for more critical applications, for example automated features. The architecture structure makes it easier to systematically add redundancy, e.g. redundant computational nodes, redundant communication links including switches and gateways, and redundant I/O nodes.

10.2.3 **ST7.6.3 Partitioning and allocation of software**

Within this task we have together with EMC^2 partner DTU developed a method and tool for allocation and scheduling of single core ECU software to a multi core ECU.

Design and implementation of functionalities and software that can harvest the full potential of multicore ECUs is a non-trivial task, especially in the context of AUTOSAR. This is not only applicable to the potential future software but also to the legacy code base. To this end, UC 7.6 aims at exploring concepts, methods and tools for efficient partitioning, allocation and mapping of functionalities as well as software across the multicore ECUs at the E/E architecture level and across the processor cores within a particular multicore-based ECU.

In line with this, ST7.6.3 has been investigating, on a conceptual level, design-space-exploration for task-allocation and scheduling. Furthermore, UC7.6 was investigating and developing methodologies for software-to-core allocation and task/runnable scheduling (i.e. ECU-level) as well as software-to-ECU allocation and network-configuration (system-level) in a near-optimal manner. The methodology explores the design-space in a systematic manner, while considering user-given constraints.

The activities of ST7.6.3 consider developing an offline algorithm to map multiple tasks of an automotive application on a multicore ECU. The tasks of the application have precedence/dependency constraints, hard/soft real-time constraints (e.g., periods, deadline), and can also have mixed-criticality (i.e., ASIL) constraints. The algorithm takes the tasks, the constraints, and the model of the hardware as input. The output of the algorithm is the allocation and schedule of tasks in each core such that all the constraints are met. The current approach is implemented in a tool from DTU and has been applied on legacy software from an existing single-core ECU in order to partition the software for a multicore solution. The tool is currently being extended by DTU in order to encompass the necessary assumptions and features to allow it to be fully compliant in an industrial setting.

Visualization of software architectural design and implementation at different abstraction levels (e.g., logical design component, software component, task, runnable) may potentially assist in identifying the possible improvement areas while developing software for multicore ECUs. This can eventually lead to more multicore-friendly, i.e., exploitation of the potential of multicores, software. To this end, ST7.6.3 investigates, adapts and develops methods and tools to analyze software. A visualization tool has been developed that is capable of showing different dependency views for the software. The tool operates on

Volvo specific data formats used in product development which facilitates integration of the tool in the current build environment and ensures industrial uptake.

Another relevant task is to develop as well as adapt method and tool support to merge the functionalities of multiple existing single-core ECUs into a single multicore ECU. This can reduce not only the number of ECUs but also the variants across the ECUs in the next-generation E/E architecture.

See Figure 73 for an overview of the AUTOMAP tool.

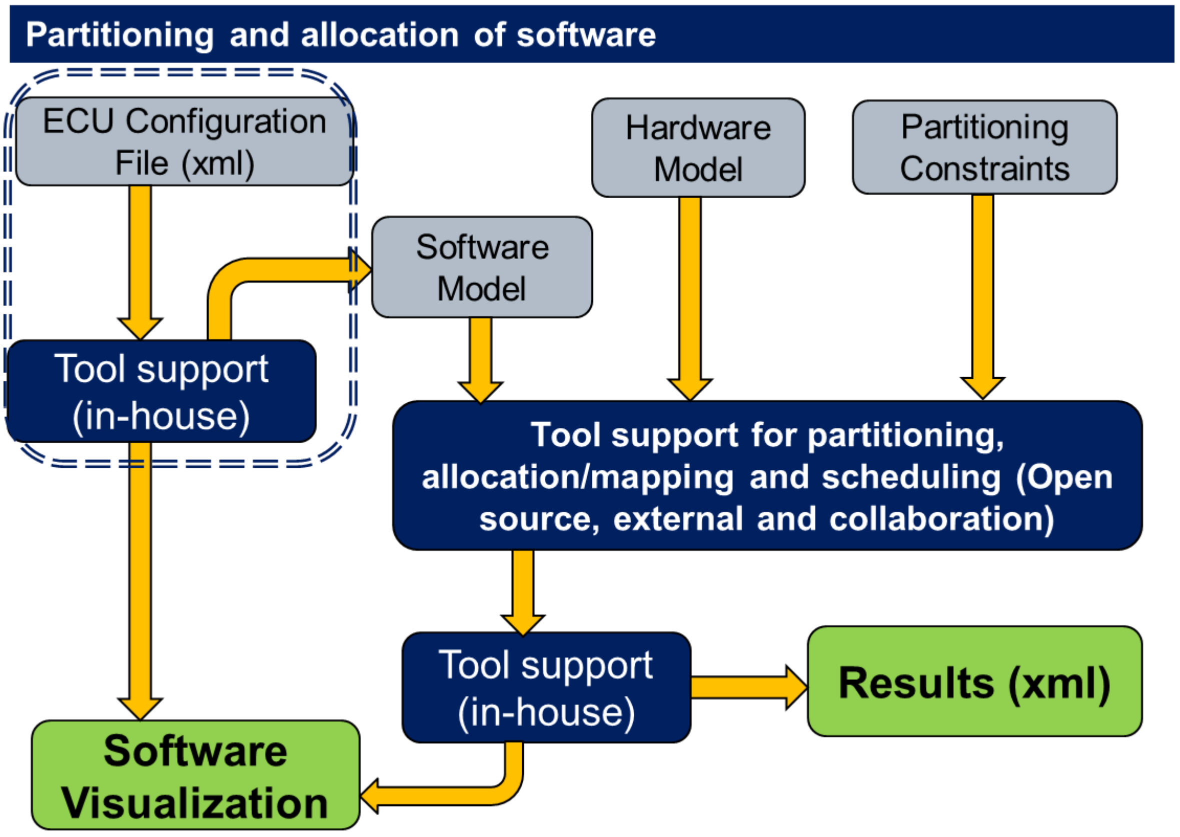

Partitioning and allocation of software
ECU Configuration File (xml)
Hardware Model
Partitioning Constraints
Software Model
Tool support (in-house)
Tool support for partitioning, allocation/mapping and scheduling (Open source, external and collaboration)
Tool support (in-house)
Results (xml)
Software Visualization

Figure 73: AUTOMAP overview

Complementary, VIF has focused on developing methods for scheduling mixed-critical software-components on a partitioned multi-core platform. The methodology contains the following:

First, we have developed an enhanced schedulability-analysis. It is based on response-time analysis (RTA), where the worst-case response-time of each task is calculated.

$$R_i^{n+1} = J_i + B_i + C_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{R_i^n + J_k}{T_k} \right\rceil \cdot C_k$$

Equation 1:

This schedulability-analysis has been extended by **fault-injection** capabilities. This allows us to evaluate the system's schedulability under erroneous behavior. We can inject 2 types of timing-related faults:

- (1) prolonged task's execution time: Due to an error, the execution-time of a task is longer than estimated for "normal behaviour". This effects the term C.
- (2) shortened periodic task's triggering: Due to an error, a task is triggered more frequeltly than estimated for "normal behaviour". This effects the term T.
  These faults can be injected to any task in the system, can be combined in any way (single fault, multiple faults), and be aligned with criticalities.

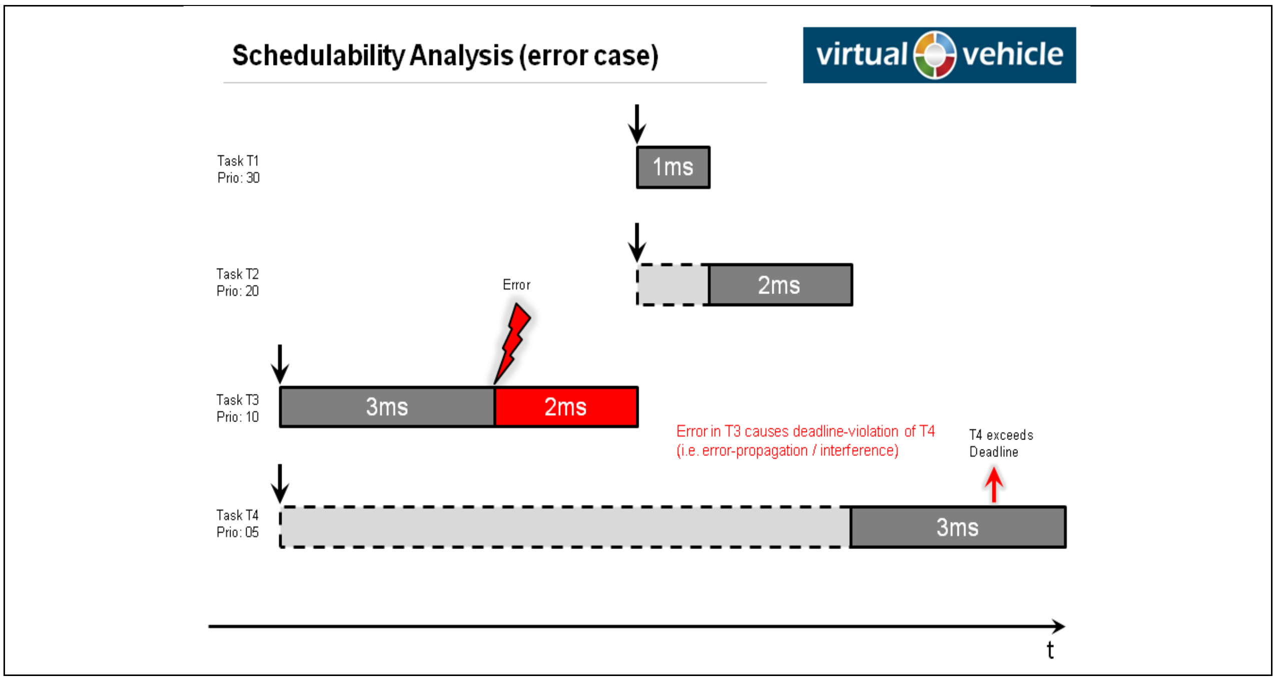

Schedulability Analysis (error case)
virtual vehicle
Task T1
Prio: 30
1ms
Task T2
Prio: 20
2ms
Error
Task T3
Prio: 10
3ms
2ms
Error in T3 causes deadline-violation of T4
(i.e. error-propagation / interference)
T4 exceeds
Deadline
Task T4
Prio: 05
3ms
t

Figure 74: Schedulability-analysis with fault-injection:
By injecting a "prolonged execution time" fault into task T3 we can analyse how this fault propagates through the system. We find that this fault causes a deadline violation in task T4.

Second, we have extended the schedulability-analysis with **timing-protection** mechanisms:

- (1) execution-time monitoring: If the execution-time of a task exceeds a pre-defined "execution-time budget", the task is stopped.
- (2) period / inter-arrival rate monitoring: If a task is triggered too frequently, the task is delayed until a pre-defined value. This is similar to "traffic shaping".

These protection-mechanisms are in alignment with the AUTOSAR standard (i.e. AUTOSAR timing-protection, v.4.x). This allows us to evaluate the system's schedulability under erroneous behavior and active timing-monitoring.
It also allows us to evaluate the effectiveness of the timing-protection mechanisms, and to check if the timing-protection-parameters have been set correctly.

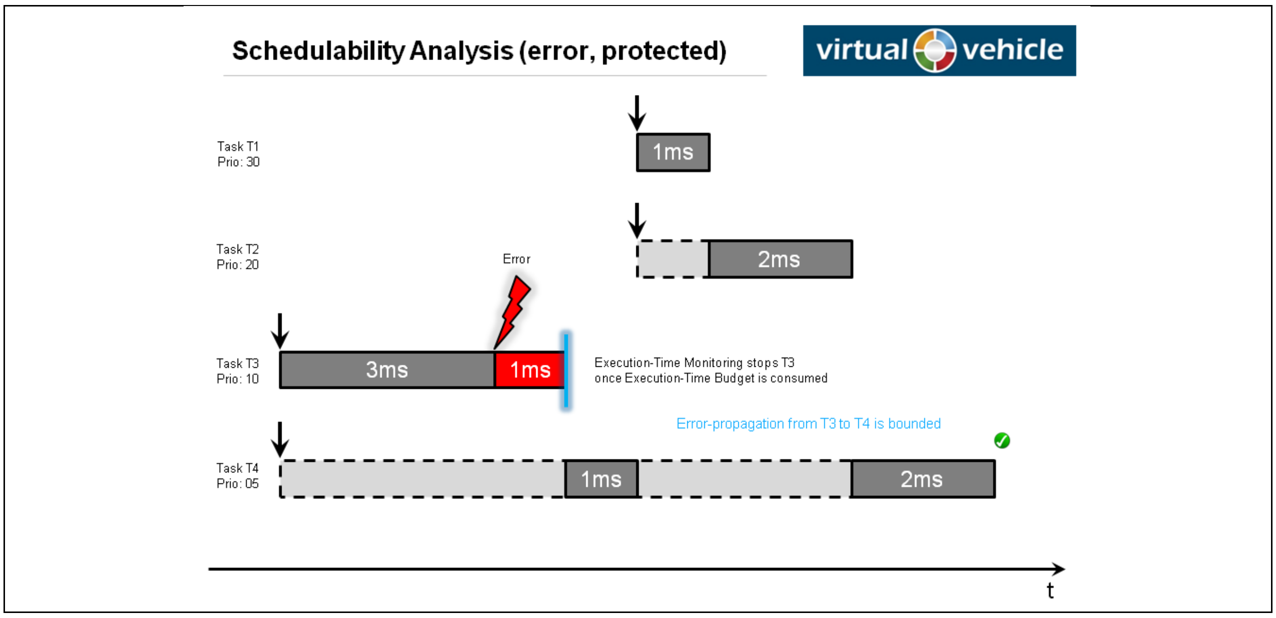

Schedulability Analysis (error, protected)
virtual vehicle
Task T1
Prio: 30
1ms
Task T2
Prio: 20
2ms
Error
Task T3
Prio: 10
3ms
1ms
Execution-Time Monitoring stops T3
once Execution-Time Budget is consumed
Error-propagation from T3 to T4 is bounded
Task T4
Prio: 05
1ms
2ms
t

Figure 75: Schedulability-analysis with fault-injection and AUTOSAR's timing-protection mechanisms:

Although a "prolonged execution time" fault is injected into task T3, the fault does not propagate through the system, and does not cause any deadline violations. This is due the active "execution-time monitoring" timing-protection for task T3. After a pre-defined "execution-time budget" is exceeded, the scheduler/OS halts task T3.

Further, we have developed a methodology for finding appropriate timing-protection-parameters and task-priorities in an automated manner. These take into account the task's criticality (i.e. safety-integrity-levels, ASIL), such that no unbounded timing-interference can occur (i.e. there is a safe upper bound for timing-interference). This way we can satisfy "freedom-from-interference" (as demanded by safety-norms, such as ISO26262). Currently, VIF evaluates the developed methodologies on some synthetic examples.

10.2.4 ST7.6.4 Dependability aspects

UC7.6 has investigated methodologies to achieve compliance with safety/dependability requirements, with special focus set on the aspect of "freedom from interference" from the ISO 26262 [2]. Virtualization and related technologies has been investigated and adapted to the context of the automotive E/E systems to facilitate ensuring freedom from interferences.

In relation to the ISO 26262, the work included investigating how to fulfill functional safety requirements of ISO26262 using robustness testing techniques.

Technologies and solutions to support co-location of multiple operating systems on a single multicore CPU has been investigated and adapteded to allow integration of safety-critical and non-critical applications on the same CPU. In particular, the issues related to the co-location of other OS, for examples, Linux and along with AUTOSAR OS using Xen hypervisor has been investigated.

The conclusion of the investigation of using Xen open source hypervisor to run AUTOSAR OS and Linux in the same ECU is that Xen for ARM CPUs is not seen as mature enough to be used in automotive ECUs at the time of the investigation, especially in areas Fast bootup, Supporting documentation and Easy debugging.

ViF (with support from Volvo and Arccore) has developed metrics for analysing if (or to which degree) an ECU-configuration satisfies "freedom from interference" requirements (according to ISO 26262). This is done at different "levels" (runnables, tasks, partitions, cores). The metrics can be utilized for (a) evaluating design-alternatives, and lateron for (b) design-space exploration.

10.2.5 ST7.6.5 Service-oriented Architecture

The main goal here was to investigate the implications of moving towards a Service Oriented Architecture for a future Reference Architecture for a Product Line of (embedded) software and electronic artifacts supporting several truck families.

Most of this electronic and software in trucks has not been recognized as a meaning by itself as compared to software-made applications like MS Word, Google Chrome, Rhapsody modeling tool, etc. Hence electronics and software in trucks is often referred to as an "embedded system" or several cooperating "embedded systems". Historically embedded system has been very closed systems with few, primitive and commonly proprietary mechanisms for environment interactions.
However, with the rise of interconnected devices and machines, historically closed embedded systems, and smart analytics, we are experiencing a technological shift not seen since the Internet revolution of the 1980s-90s. With the rise of the Internet of Things (IoT), truck industry sees improved productivity, major cost savings, and streamlined processes. Traditional "embedded systems" become heavily interweaved and interconnected with its environment following "open" standards and the border between an "embedded system" and other internet-based applications is blurred, e.g. controlling the cooling temperature of a truck cab while parked can equally well be managed in cloud-based application software as in an in-vehicle application.

The prevailing solution for in-vehicle software is based on AUTOSAR. Although some aims overlap, for example that the allocation of software should be possible to make at a late stage. To go from AUTOSAR to SoA constitutes a paradigm shift from design time engineering of a static allocation to a run-time scalability of hardware and software components. The current E/E system architectures in automotive industry are characterized by a large number ECUs that are interconnected by CAN buses. Both AUTOSAR and SoA define interfaces between software components instead of a CAN hardware specific communication matrix. The relative benefit of this approach increases with the size complexity of the application and in automotive systems the number of software functions is steadily increasing.

The drawback of a flexible computer system approach such as SOA is the problem of assuring dependability, i.e. to make sure it is predictable safe and secure. Instead of pure design time analysis and testing carried out with a statically configured system, run-time supervision is a key functionality but much theoretical work needs to be put into this before it can be applied to safety-critical features.

Vehicle features can be split into a set of domains. Not all domains of the vehicle are safety-critical. In the non-safety-critical infotainment and telematics domains the life-time of technical solutions is more similar to consumer electronics than the life-time of for example the safety-critical service brake system domain. During the life-time of the commercial vehicle, it would probably be a possibility to offer customers hardware and software upgrades a couple of times. The ability to do this is greatly facilitated by a flexible computer system based on plug-and-play techniques such as found as a cornerstone of SOA. The business incentives and thus the likelihood of near future introduction of a SOA concept are larger in the infotainment and telematics domains. In the infotainment and telematics domains, system wide features mentioned in the description of work such as external network access, secondary storage (repository) and optimization of e.g. route selection could be located.

This overall research question can be divided into the following sub-questions:

1. Does a Service Oriented Architecture change how one needs to look upon architectures in the automotive domain today, which today is very mechanic, geometric and physical component-oriented?
2. What is a “service” for a “truck” according to a Service Oriented Architecture?
3. What is a “proper” granularity (size) of a “service” for a “truck”?
4. Does a Service Oriented Architecture have any (major) impact on how one need to fundamentally organize assets of the product line for trucks?
5. Does a Service Oriented Architecture rely upon the existence of some underlying (basic) techniques and/or architecture styles/patterns, e.g. the Internet Protocol, Client-Server oriented, etc.?

The approach selected is to target a specific feature called “Compartment Climate Service” in a truck that concentrates is answering most of the research questions above. This climate control is used to warm or cool vehicle interior air, and to defrost the windscreen and the windows. It is connected to the electrical architecture and cooling water circuit of the vehicle.

Chalmers considers a dependable system which is able to (i) degrade its functionality dropping low priority tasks or reducing their functionality/precision/quality of results, and (ii) reconfigurable processing cores which may be able to recover from permanent faults (at the cost of efficiency/performance). To this end, Chalmers develops runtime algorithms which make use of the adaptability of a modern MPSoC to work around permanent faults with the objective to (i) always respect hard constraints, like realtime deadlines, critical functionality and (ii) to optimize an objective function which reflects the system’s Quality of Service (QoS) and is derived from the system’s performance, energy efficiency and achieved functionality.
Working towards this direction and having in mind Volvo’s power-train application, Chalmers has developed a number of different runtime algorithms and performed comparisons between them in terms of solution accuracy (compared with the optimal value of the objective function in each case) and execution time (which shows how long a system has to remain idle for the new solution to be computed). The following three algorithms were developed:

1) A **custom runtime heuristic**, first seeking an incremental solution (a system configuration resulting by changing only a few elements of the previous configuration) and then using a precomputed solution (a pessimistically inefficient, but working, solution tailored to the particular failure scenario).
2) **Simulated annealing**, which is a randomized algorithm, moving inside the search space, occasionally accepting solutions that are worse than the best known so far, for the sake of introducing diversity and escaping local maxima. As the algorithm runs, a factor called "entropy" is gradually reduced, limiting this tendency to jump around and focusing more on improving the so far best solution.
3) **Genetic algorithm**, which maintains a "population" of solutions and performing a crossover operation between pairs of them to create new ones, called "offsprings". After a number of generations, the best solution encountered in the population is returned.

The experiments with the three different algorithms yielded so far the following (preliminary) results:

- The incremental nature of algorithm (1) results to better QoS during the first 30-50% of the system lifetime, since this algorithm is able to find a system configuration very close to the initial, optimal one, whereas algorithms (2) and (3) result to a severe drop of QoS when the first fault takes place.
- After the first fault, algorithms (2) and (3) maintain the QoS better than algorithm (1), resulting in a QoS higher by 4-8 percentiles of the scale, close to the end of the system lifetime.
- Algorithm (1) takes less time to execute, being more than one order of magnitude faster than algorithm (2) and roughly 2 orders of magnitude faster than algorithm (3).

This task investigates the suitability of SoA and run-time optimizations for real-time safety-related automotive systems along with identification of associated challenges and implications.

The SoA applied here follows closely the design pattern proposed in WP1, e.g the common definition of metadata. For the investigation of applicability, a specific vehcile feature "Compartment Climate" in its simplest form is selected.

System overview:
This compartment climate system is used to warm or cool vehicle interior air, and to defrost the windscreen and the windows. It is connected to the electrical architecture and cooling water circuit of the vehicle. The main functions are controlling the:

- Temperature
- Air distribution
- Blower
- AC mode
- Recirculation air

The error, fault handlings, fallback scenarios, etc are ignored.
Handling climate control from bunk is ignored.
Displaying statues to the driver interface display is ignored. However, the indication about climate control information to the driver is taken into account
Interactions with battery is ignored
Interactions with washer wiper is ignored
All variations are ignored e.g. variations due to brand or segment.
The focus of this simple compartment climate is providing an option to set a specific temperature in the cabin based on the temperature control functionality specified in an SAE publication, see [2].

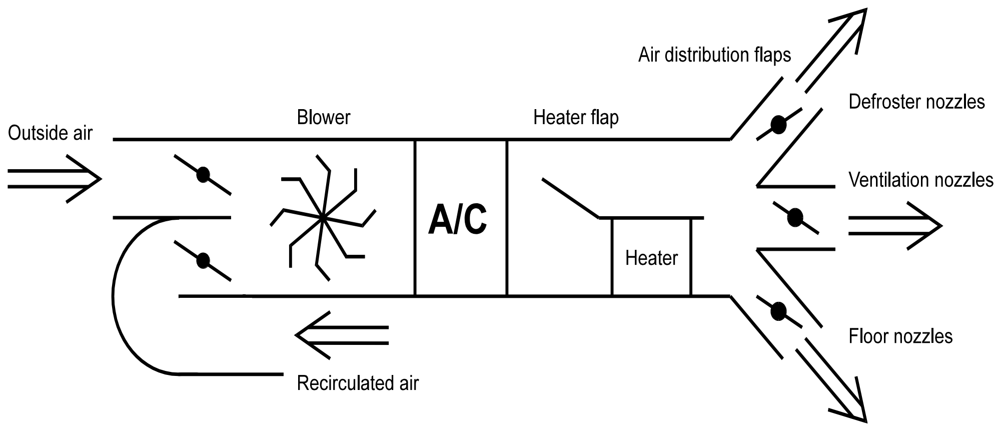


Figure 76: Climate control system from a control engineer's perspective

Representation using SoaML:
SoaML is an OMG standard with small set of extensions to UML2 to support SOA modeling. It adds several stereotypes necessary for specifics of service oriented architecture, such as participant, service interface, provided interface, required interface, etc. However, to systematically use this in a development process requires a common understanding of mapping from SoaML design artifacts to the commonly known (to us) implementation artifacts such as class, object, operations, interface etc. Also, SoaML gives us the bits and pieces to solve the representation puzzle, but leaves many options open for us to define our own way of using it. There are different tool supports available, for example, Enterprise Architect, Visual paradigm etc.
The approach in SOA is that people, organization and systems provide service to each other. The services gets something done for the consumer who doesn't want to do it itself and even without knowing how it is done. A service is value delivered through a well-defined interface. The following main artifacts are the core of SoaML

Participants: Specific entities that provide or use services. It can represent people, organization and any information system components. It may provide any number of services and consume any number of services.

Ports: Participants provide or consume services via ports. It is part of the participant that is the interaction point for a service. A port where a service is offered is called "Service" port and a port where a service is consumed is called a "Request" port.

Capabilities: Participants that provide a service must have a capability to provide it. Different providers may have different capabilities to provide the same service.

Service description: Description of how the participant interacts to provide or use a service is encapsulated in a specification for the service. There are three ways to specify a service interaction:

- UML Interface
- ServiceInterface
- ServiceContract

It seems the reason for having these three ways is because each one is suitable depending on the complexity of a service. However, each one results in interfaces and behaviors that define how a participant will provide or use a service through ports. The service description does not specify anything about how it is implemented. This separation of concern is fundamental to SOA.

The three approaches of specifying a service is summarized as:

UML Interface based: This can be used for a one-way interaction provided by a participant on a port represented as a UML interface. The participant receives operations on this port and may provide results to the caller. This kind of one-way interface can be used with "anonymous" callers and the participant makes no assumptions about the caller or the choreography of the service. The one-way service corresponds most directly to simpler "RPC style web services" as well as many "OO" programming language objects. Simple interfaces are often used to expose the "raw" capability of existing systems or to define simpler services that have no protocols. Simple interfaces are the degenerate case of both the ServiceInterface and ServiceContract where the service is unidirectional - the consumer calls operations on the provider - the provider doesn't callback the consumer and may not even know who the consumer is.

ServiceInterface based: It allows bi-directional services where there are "callbacks" from the provider to the consumer as part of the conversation. It is defined in terms of the provider of the service and specifies the interface that the provider offers as well as the interface, if any, it expects from the consumer. A consumer of a service specifies the service interface they require using a request port. The ServiceInterface is the type of a "Service" port on a provider and the type of a "Request" port on the consumer. Compatibility of service interfaces determines whether these agreements are consistent and can therefore be connected. The ServiceInterface approach is most applicable where existing capabilities are directly exposed as services or in situations that involve one or two parties in the service protocol.
ServiceContract based: A service contract approach defines service specifications (the service contract) that specify how providers and consumers work together to exchange value. The service contract approach defines the roles each participant plays in the service (such as provider and consumer) and the interfaces they implement to play that role in that service. These interfaces are then the types of ports on the participant, which obligates the participant to be able to play that role in that service contract. The service contract represents an agreement between the parties for how the service is to be provided and consumed. This agreement includes the interfaces, choreography, and any other terms and conditions. In summary, the service contract approach is based on defining the service contract "in the middle" (between the parties) and having the ends (the participants) agree to their part in that contract, or adapt to it. The ServiceContract approach is most applicable where an enterprise, community SOA architecture or choreography is defined and then services built or adapted to work within that architecture, or when there are more than two parties involved in the service.

The fundamental differences between the contract and interface based approaches is whether the interaction between participants are defined separately from the participants (in a ServiceContract based) or individually on each participant's service and request interface (in an interface based).

The figure below gives you an overview of the modeling of compartment climate system using SoAML in Enterprise Architect.

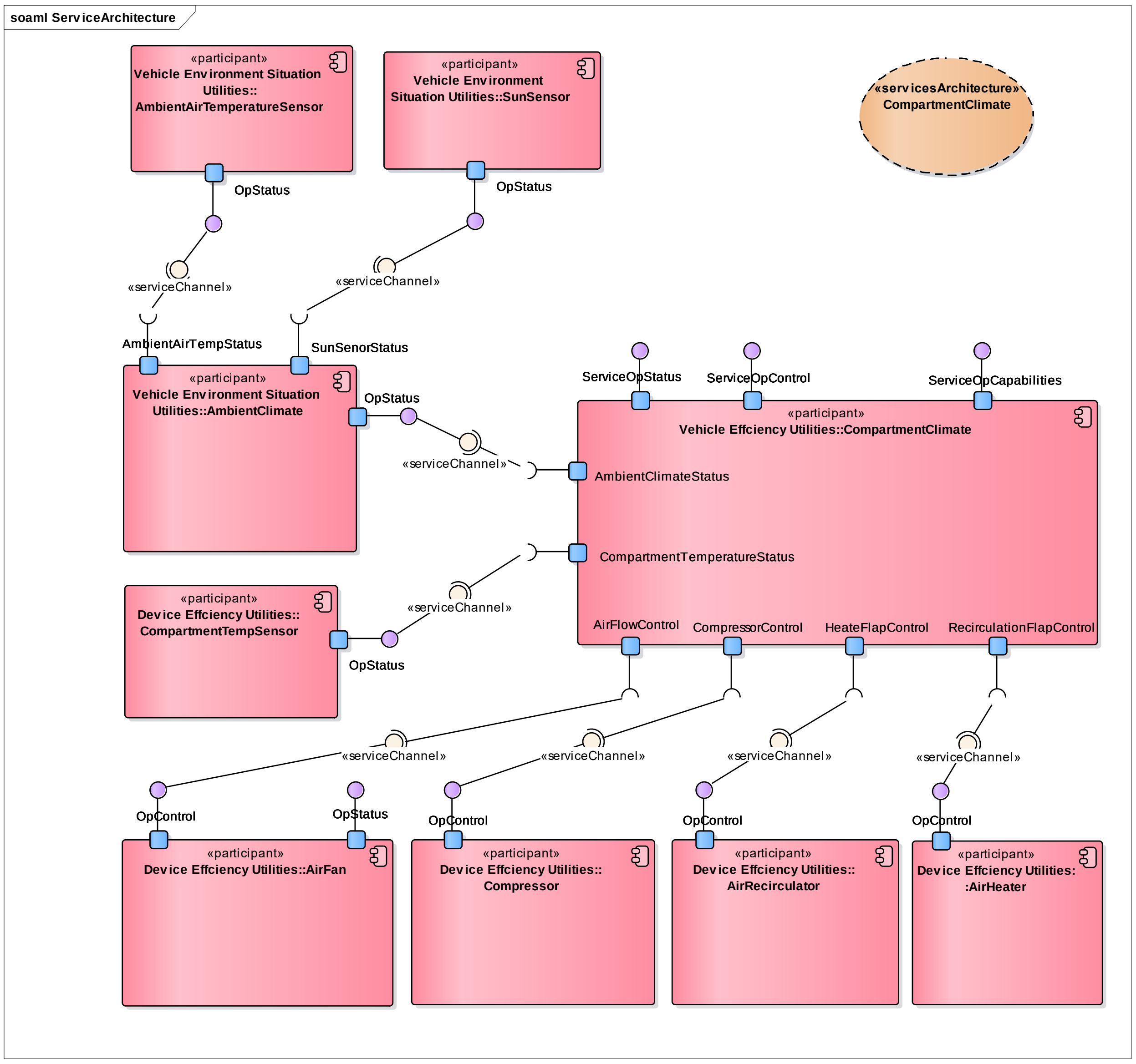

soaml ServiceArchitecture
«participant» Vehicle Environment Situation Utilities:: AmbientAirTemperatureSensor
«participant» Vehicle Environment Situation Utilities::SunSensor
«servicesArchitecture» CompartmentClimate
OpStatus
OpStatus
«serviceChannel»
«serviceChannel»
AmbientAirTempStatus
SunSenorStatus
«participant» Vehicle Environment Situation Utilities::AmbientClimate
OpStatus
«serviceChannel»
ServiceOpStatus
ServiceOpControl
ServiceOpCapabilities
«participant» Vehicle Effciency Utilities::CompartmentClimate
AmbientClimateStatus
CompartmentTemperatureStatus
AirFlowControl
CompressorControl
HeateFlapControl
RecirculationFlapControl
«participant» Device Effciency Utilities:: CompartmentTempSensor
«serviceChannel»
OpStatus
«serviceChannel»
«serviceChannel»
«serviceChannel»
«serviceChannel»
OpControl
OpStatus
OpControl
OpControl
OpControl
«participant» Device Effciency Utilities::AirFan
«participant» Device Effciency Utilities:: Compressor
«participant» Device Effciency Utilities:: AirRecirculator
«participant» Device Effciency Utilities: :AirHeater

Figure 77: Modelling of Compartment Climate system using SoaML

Hardware setup for the demonstration:
The complete test setup is shown in figure below.

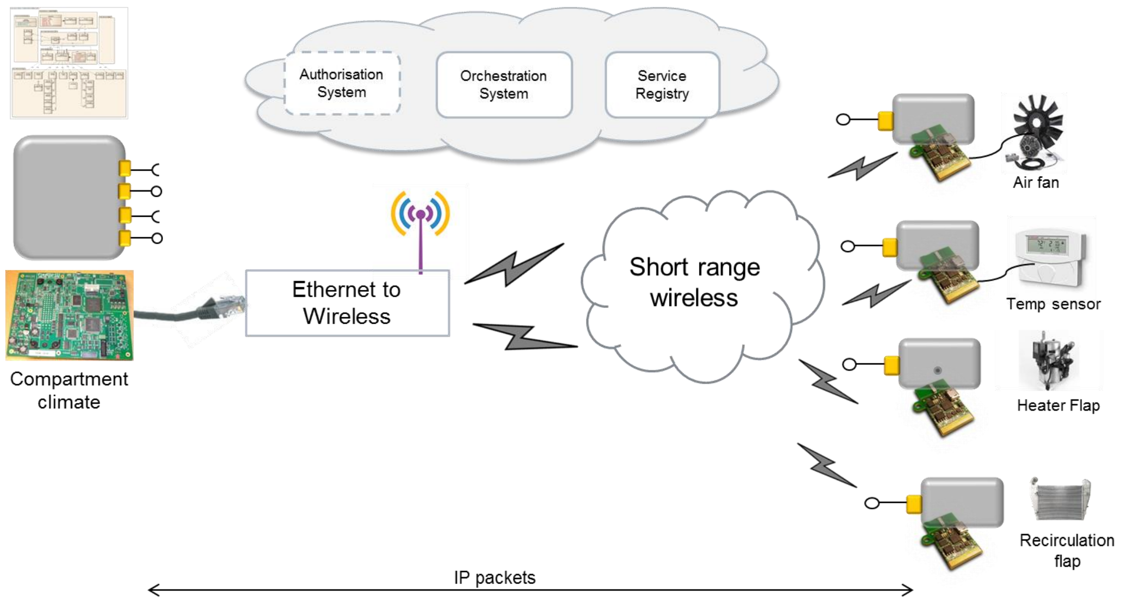


Figure 78: Hardware setup for demonstration of Compartment Climate system

10.2.5.1 Runtime optimizations for Graceful system degradation

In D7.14, Chalmers described their work on T7.6 (ST7.6.5 Service-oriented Architecture) adapting various existing algorithms (simulated annealing and genetic algorithms) and creating new custom heuristics to efficiently degrade a system gracefully when permanent faults occur. In this work, Chalmers considered a dependable system which is able to (i) degrade its functionality dropping low priority tasks or reducing their functionality/precision/quality of results, and (ii) reconfigurable processing cores which may be able to recover from permanent faults (at the cost of efficiency/performance). The developed runtime algorithms had in mind Volvo's power-train application and make use of the adaptability of a modern MPSoC to work around permanent faults with the objective to (i) always respect hard constraints, like realtime deadlines, critical functionality and (ii) to optimize an objective function which reflects the system's Quality of Service (QoS) and is derived from the system's performance, energy efficiency and achieved functionality.
Three algorithms were developed in period 2 and subsequently evaluated in period 3. The outcome of this work was published in:

"Runtime Management of Adaptive MPSoCs for Graceful Degradation", Int. Conf. on Compilers, Architectures and Synthesis For Embedded Systems (CASES), S. Tzilis, I. Sourdis, V. Vasilikos, D. Rodopoulos and D. Soudris, Embedded Systems week, Pittsburg, PA, USA, Oct. 2-7, 2016.

We first briefly describe the three algorithms and then our evaluation results:

a) A custom runtime heuristic, first seeking an incremental solution (a system configuration resulting by changing only a few elements of the previous configuration) and then using a precomputed solution (a pessimistically inefficient, but working, solution tailored to the particular failure scenario).
b) Simulated annealing, which is a randomized algorithm, moving inside the search space, occasionally accepting solutions that are worse than the best known so far, for the sake of introducing diversity and escaping local maxima. As the algorithm runs, a factor called "entropy" is gradually reduced, limiting this tendency to jump around and focusing more on improving the so far best solution.
c) Genetic algorithm, which maintains a "population" of solutions and performing a crossover operation between pairs of them to create new ones, called "offsprings". After a number of generations, the best solution encountered in the population is returned.

Evaluation results:
In order to evaluate our strategy, we set up simulations of numerous failure scenarios. The simulations were based on a system consisting of 4 components, 2 of which lightweight cores and 2 more powerful cores. Each component has 4 possible configurations: one fully functional, 2 different degraded and one failing. The workload running on the MPSoC is a use case that could run on a passenger vehicle, consisting of essential automotive tasks and optional features for the passengers' comfort and entertainment taken from the MiBench. The QoS metric, attempted to be maximized by the system optimization algorithms when parts of the system fail, reflects (i) the functionality supported by the system, (ii) its performance, and (iii) energy efficiency.

The simulations were set up based on a number of parameters, below. We have defined 72 failure scenarios in total, by varying the following parameters. The base fault rate, λ, between 0.004 to 0.013 faults/day. The probability that a fault will be fatal, P(fatal). A fatal fault does not allow the affected component to degrade gracefully, but incapacitates it completely. This emulates the possibility that faults will be either too severe, or too inconveniently placed for the affected component to be salvaged, despite the reconfiguration capabilities. The spatial fault distribution. We tried uniform random (all components equally susceptible) and the "weak component" model, according to which one random component is much more vulnerable than the rest. The behavior of the fault rate, λ, in time. We tried constant fault rate as well as fault rate peaking around random moments in the system lifetime (corresponding to critical periods of time, during which faults are more probable to happen).

There are 72 combinations of values for the above 4 parameters. We performed 5 simulations for each combination, resulting to a total of 360 simulations. Each simulation was terminated either when the GD manager could not provide a valid system configuration or after 1000 days. Figure 79 illustrates the results of the simulations in terms of quality of service (QoS) on the leftmost graphs and percentage of systems still working on the rightmost graphs, over time. Each line in the graphs corresponds to the average value of a subset of the experiments, characterized by a fixed value for one of the above parameters. For example, the top graphs depict results for different values of the base fault rate, λ. The line named "All" on each graph corresponds to the average of all 360 experiments. Again, for example, on the topmost graphs, the line labeled "All" corresponds to the average of all experiments, regardless of base fault rate.

The impact of the fault rate as well as the probability of fatal fault on the results is clear, as was expected: higher values for λ and P(fatal) result to more rapid degradation. On the other hand, the weak component model makes small difference (for the better), compared to the uniform random fault distribution. Presumably, it is slightly easier to deal with multiple faults on the same component; after the first fault, its duties are reduced, thus subsequent faults on it result in minor changes. Lastly, during runs with the fault rate peaking at random time periods, the GD manager does slightly worse than with constant fault rate, since the peaks slightly increase the expected number of total faults.

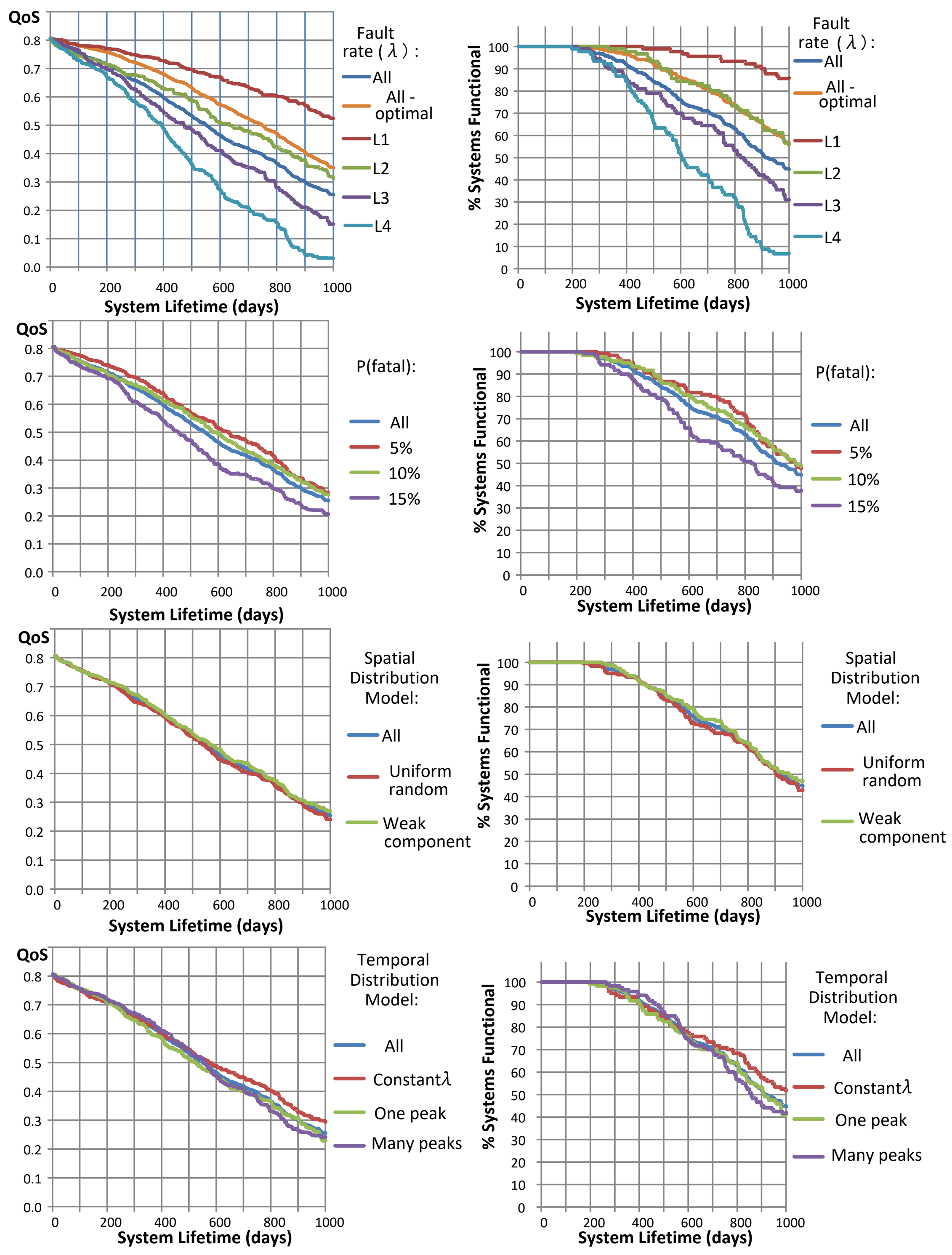
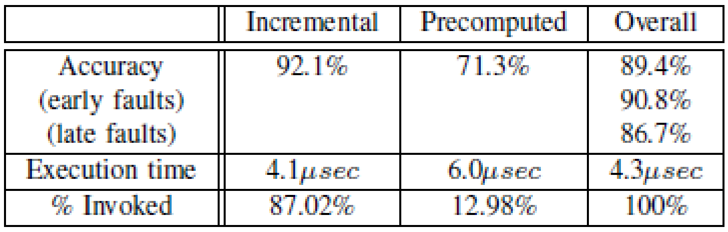


Figure 79: The average QoS (left) and percentage of systems still working (right) over time. Different lines correspond to different fault rate (top), fatal probability (top-middle), spatial fault distribution (bottom-middle) and behavior of fault rate in time (bottom). Each graph contains a line (labeled "All") for the average of all experiments. The leftmost graphs also contain the results for the same system, if the optimal solution was always used (labeled "All-optimal").

It is also interesting to compare the leftmost graphs with the rightmost ones. Considering all 360 simulations as a whole, the distribution of faults in time is quite uniform. This results in the QoS having a roughly linear behavior in time as seen in the top graphs. The GD manager does, thus, a good job in keeping the impact of faults constant, even when they stack up. Additionally, as is visible on the bottom graphs, there is a significant period of time (roughly 20% of the total) during which almost no systems fail. After that, the % of functional systems drops in an almost linear fashion. Roughly 98% of systems work after 25% of the maximum lifetime, 83% after 50% and 67% of systems after 75%. This is the other aspect of the benefits a runtime GD manager provides: Prolonging chip lifetime, both in terms of full-fledged function and in terms of minimum acceptable service.



	Incremental	Precomputed	Overall
Accuracy	92.1%	71.3%	89.4%
(early faults)			90.8%
(late faults)			86.7%
Execution time	4.1μsec	6.0μsec	4.3μsec
% Invoked	87.02%	12.98%	100%

Table 26: Accuracy and execution time statistics for the GD algorithm.

Finally, we evaluate how close to the optimal are the solutions produced by the heuristic. On the leftmost graphs of Table 26, we include the behavior of the same system, assuming the new configuration was calculated by exhaustive search in place of the heuristic (illustrated by the line named “All-optimal”). Lines “All” and “All-optimal” naturally start from the same QoS value for 0 days. The optimal QoS gradually gets better than the achieved one over time, reaching a difference of 0.1 after 600 days. This difference stays the same for the rest of the systems' lifetime. As listed in Table 26, the heuristic produces on average solutions delivering QoS 89% as high as the guaranteed-optimal exhaustive search. This is much lower (71%) when the precomputed partial solution is used. This is also to be expected: the precomputed solution is pessimistic (thus suboptimal), because it is the last chance of the GD manager to produce a non-failing configuration. Fortunately, only 13% of the solutions are based on precomputation. Furthermore, for invocations of the heuristic within the first half of simulation time, accuracy is slightly higher (90.8%) than the second half (86.7%), a difference attributed to the fact that later calls of the heuristic start from a sub-optimal configuration that was produced by an earlier call. Lastly, the algorithm takes on average 4.3µsec to run on a LEON2 core. As expected, this time is significantly (50%) longer if the second part (precomputed solution) of the algorithm needs to be invoked.

10.2.6 **ST7.6.6 Demonstration and evaluation**

Demonstrators from this use case are described in the following sections. The demonstrators have been used for evaluation and dissemination at partner’s site, at project conferences and at external conferences.

10.2.6.1 AUTOMAP

The method and tool (AUTOMAP) for allocation and scheduling of single core SW on multicore ECUs has been developed in cooperation between VOLVO and DTU. The demonstrator was shown in the project conference 2015 in Vienna and in the second project review 2016 in Gothenburg.
The tool has been developed based on internal VOLVO needs using real examples from product development. The evaluation and validation of the tool has been started internally in VOLVO. The tool is tailored for the internal VOLVO needs but formal results from the validation are not available for publication at this point in time. See more information in section 10.2.3.

10.2.6.2 Service-oriented Architecture

The demonstrator showing an implementation of the SoA framework from WP1 has been developed in cooperation between VOLVO and LTU. The demonstrator was shown in the second project review 2016 in Gothenburg and at the Artemis Technology Conference 2016 in Madrid.
The demonstrator has been part of the evaluation of the feasibility of using SoA in the embedded truck architecture. See more information in section 10.2.5.

10.2.6.3 Architecture concepts

A prototype is under development internally at VOLVO. The prototype will be extend with time and is used for evaluation of different kinds of concepts for future embedded truck E/E architecture and is also used for internal dissemination. See more information in section 10.2.2.

The demonstrators and evaluation for the ArcCore activities are described in the following sub sections.

10.2.6.4 Arctic Core on Xen Hypervisor

In the first review meeting in EMC^2, Arccore showed a prototype of the Arctic Core running on Xen. To achieve this, changes were needed in both the OS and the MCAL of the Arctic Core AUTOSAR stack.

The demonstrator showed that multiple instances of Arctic Core and Ubuntu can be run side by side running on top of the Xen hypervisor. But it also showed that the usage of a hypervisor requires new knowledge and introduces more effort in the development process.

10.2.6.4.1 Hypervisor applicability within automotive applications

The software within the automotive sector is growing and the demand for advanced functionality such as cloud integration, video-processing, sensor fusion are being introduced with the integration of smarter cars. The automotive microcontrollers with multiple cores are increasing and getting a larger market share for each year. The demand for more complex applications and electronic control units using multiple OS´s are also growing within the automotive sector as the more advanced tasks also requires more specialization. The number of ECU´s within the vehicle are also growing with the demands of smarter diagnostics and increased functionality. The increased demand on performance, safety, speed and the complexity of the functionality are creating an increased demand of modularity within the software development. This modularity is partly covered by introducing Linux and Autosar. But to reduce the overall number of control units in a vehicle is becoming an identified goal for the vehicle manufacturers.
The effort to achieve functional safety according to ISO26262 can be reduced if a hypervisor is used to separate between the safety critical software and the non-safety critical software and OS´s.
The hypervisor can also function as an enabler for increased security by separating the security critical software from the non-security critical software. The hypervisor can also be used to separate between connected applications and the vehicle controls.
Hypervisors can help create all of the incentives mentioned above. And many of them are also included in the high level requirements within the work package, and especially HL-REQ-WP07-032. The hypervisor helps in allocating a multicore processor using a variety of OS´s and also integrates with bare metal code while still being both safe and predictable. As a hypervisor that is developed according to ISO26262 can help to guarantee freedom from interference this means that a linux distribution can run in the same microcontroller as safety critical content.

10.2.6.4.2 Automotive Hypervisor needs

- Safety

ISO 26262 dictates freedom from interference as a method to separate between non-safety critical (QM) software and safety critical (ASIL A-D) software. While the hypervisor can be one of the methods to achieve freedom from interference, ISO 26262 will still put requirements on the hypervisor to be developed according to the standard. As the hypervisor will influence both the scheduling and memory separation of the user domains it can be concluded that the hypervisor needs to be developed to the standard of the highest rated ASIL level of the system.

- Boot Times

Both user experience and network performance within the vehicle requires fast boot times of the ECUs. Devices also often need to boot from power off to fulfill the requirements of energy consumption and battery life that is expected on a vehicle.

- Security

Security is a rising concern within automotive. Researchers have shown that modern car contains software that can be hacked from remote.

10.2.6.4.3 Used Tools

- Xen

Xen is a bare metal hypervisor which supports both virtualization and paravirtualization. It is open source and available under GPLv2.

While Xen was mainly aimed at high end server computers in the beginning. The Xen project is now also applied to desktop virtualization and virtualization within embedded devices. There is an automotive initiative using Xen called the Embedded and Automotive PV. Xen can been run on automotive ARM devices which are equipped with virtualization extensions. The Xen has been shown on among others the TI Jacinto 6 which is an MCU that is aimed towards the automotive market.
The Embedded and Automotive initiative has stated that they can start the Xen hypervisor in 300ms and start the Dom0 Linux in 800ms.

- Arctic Core

The Arctic Core is AUTOSAR Basic Software and RTE provided by Arccore. The Arctic Core contains the complete Autosar stack which includes OS, MCAL, BSW and RTE. The Arctic Studio provides means for the user to create their own SWC´s on top of the stack. Arctic Core is configured by the Arctic Studio, which also generates the configuration and RTE.

10.2.6.4.4 Usage

Xen allows its user to operate on virtual cores. The virtual cores can be configured to access specific memory sections and also a part of the CPU processing power. The processing power can be calculated using a selection of calculation algorithms that are provided together with Xen.

The Xen hypervisor is built up from bottom up with a Linux focus. While Xen is a bare metal hypervisor it needs to be controlled from a control Domain called Domain 0. This Domain 0 is a Linux partition in itself. On Domain 0 the Xen tool stack is run. The Xen tool stack is used to set up the other partitions, or virtual cores that they are called in Xen. The ToolStack also comes with a variety of commands to control the performance of the hypervisor and a control interface. For example, top can be run to see the performance figures divided between the virtual cores.

Each virtual core can then be treated as a physical core and operate independently of each other.
To share peripherals Xen relies on two principles. One is to open up the register sections controlling the peripheral to the virtual core that needs it, this is currently under development. The other principle is by using the Linux device tree on Dom0. The Linux device tree on domain 0 can allow the users in the other domains to connect to it and control the peripherals via the Domain 0. If a device tree is set up devices can also be masked allowing User domains to control the memory separately.

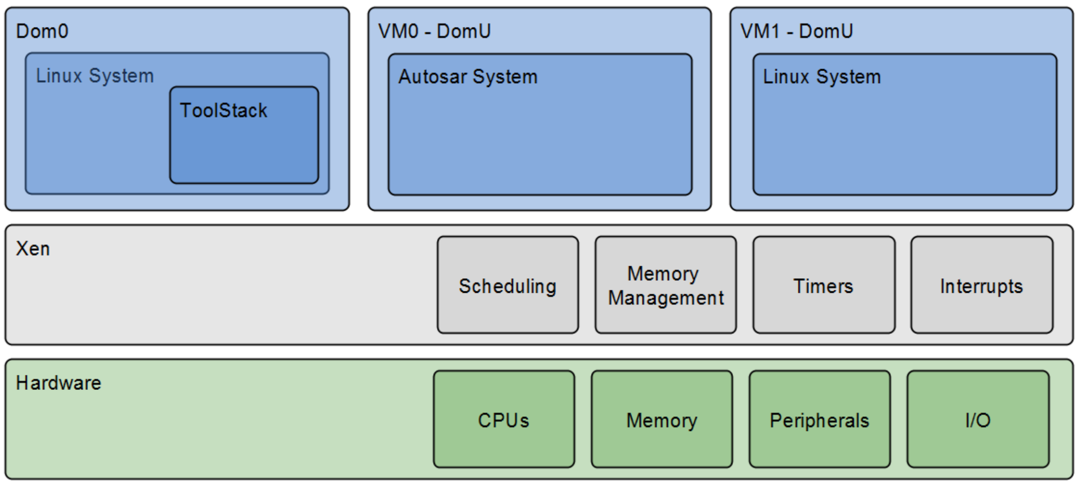


Figure 80: Xen system

Xen needs to run a control application in the Dom-0 before starting its supervised domains it has a very long boot chain. The booting of the Xen hypervisor starts with the U-Boot continues with the Dom-0 and the a virtualization toolstack. First thereafter the domain for the Autosar stack can be created and the Automotive basic functionality be started. While boot times was not in focus for the Arccore prototype, another project made by Globallogic has tried to optimize the boot times and they were still in the range of seconds and not miliseconds. A typical automotive requirement is to boot in less than 300 milliseconds.

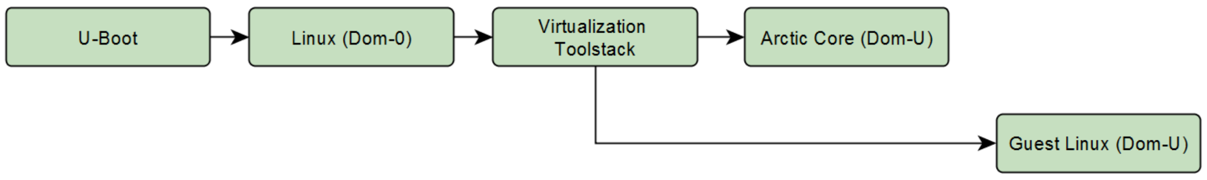


Figure 81: Picture describing the typical boot flow for creating two guest domains, one Autosar and one Linux, on the Xen hypervisor

10.2.6.4.5 Results

During the Xen project the following requirements on a hypervisor have been identified

- Fast boot sequence
- Easy debugging
- Ease of use
- Bare metal embedded orientation
- Price level
- Microcontroller support
- Developed according to ISO26262 to support Automotive Mixed Criticality (ISO26262 – ASIL D)

A successful automotive hypervisor implementation needs to address all of the above. Xen in its current state will not fulfill a sufficient amount of the above requirements. The decision was made to discontinue the Arccore Xen Demonstrator and look into other commercial hypervisors.

10.2.6.5 Active Safety Platform DrivePX2

The DrivePX2 evaluation platform is intended to provide a framework for creating applications executing a prototype ECU that is populated both with a NVIDIA Tegra SOC and an Infineon Aurix. These devices are now setting the trends within their respective domains, which are processing power and automotive functional safety. This set up enables the user to prototype powerful applications within the active safety domain. It also proves as a demonstrator for the requirements HL-REQ-WP07-023, HL-REQ-WP07-025 and HL-REQ-WP07-027.

The Aurix together with the Arccore Autosar Stack provides the means to create a unit that covers mixed criticality and allows the end user to comply to the Automotive Safety Standard ISO26262. The Arccore Autosar Platform is based on Autosar 4.2 and hosts the complete BSW stack and RTE.

The TegraB is set up with a virtual AUTOSAR platform executing in Linux. This means that it can utilize the Autosar part to handle the required native automotive functions such as diagnostics, SWDL and CAN communication while it can use the Linux platform in order to handle the ADAS functionality such as cloud, sensor fusion and connectivity requirements.

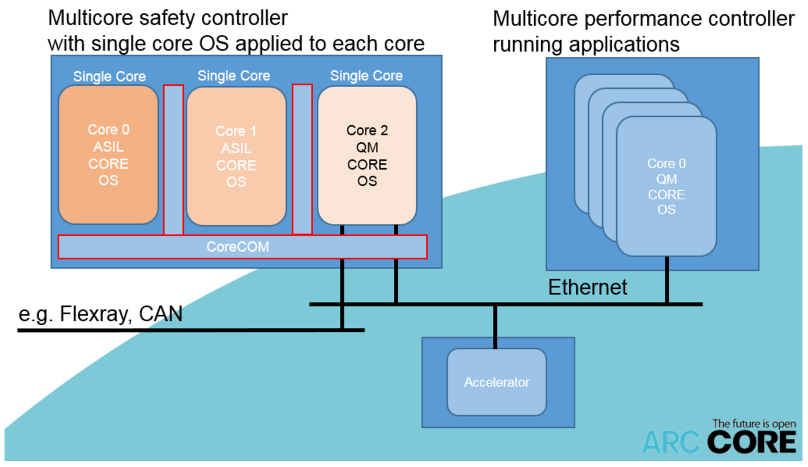


Figure 82: Multicore and multiprocessor ECU

10.2.6.5.1 Background

With the introduction of autonomous drive, safety and security aspects are of greatest importance as the cars will drive independently in traffic and the car will need to get access to online infrastructure. Furthermore, the computer power which is needed is increasing requiring solutions which use more than one MCU with multicores. Systems will require up to ASIL D safety requirements while needing a large amount of processing power. These need to be implemented compliant to ISO26262 and our prototype is using the AUTOSAR standard to demonstrate how to achieve it. Up and coming future applications for automotive software systems such as ADAS, Self driving cars and connected infrastructure will require more flexibility and functions in general.

10.2.6.5.2 Concept

Arccore demonstrates a software solution for safe multicore communication between cores that can be used to allow mixed criticality. The solution is less complex and makes it easier to fulfil ASIL requirements compared to the standard multicore AUTOSAR solution for inter-core communication. A system solution involving the complete AUTOSAR functionality and multicore specification for a complex system architecture such as the one described in the background would require a multicore OS including a multicore RTE implementation. To reduce the implementation complexity of such a system the demonstrator proposes to use multiple instances of single core OSs and RTEs instead. The cores can then communicate over a dedicated safe communication channel. The solution shall be as simple as possible, shall be developed as safety element out of context (SEooC) and shall of course fulfil the required safety integrity level.

Based on that idea further general assumptions can be derived

- Solution delivered as SEooC
- No fail operational functionality
- Same ASIL level for each core
- Core with full BSW stack available developed as QM
- Memory protection applied per core
- E2E protection applied for critical communication
- Freedom of interference with regards to timing is achieved by applying Watchdog

From these assumptions building blocks in the demonstrator including TrustedOS, TrustedCOM, TrustedWatchdog and CoreCOM can be derived to realize the approach.

The ASIL core shall contain only a minimum set of platform functionality. This limitation is to reduce the complexity and to reduce the risk in general. Furthermore, all parts on the ASIL core shall be developed according the highest ASIL level required for the applications on that core. The core shall include

- RTE (reduced, robust functionality)
- BSW (reduced, robust number of BSW modules supported including,
- TrustedOS (reduced, robust OS functionality)
- TrustedCOM (E2E, CRC)
- TrustedWdg)
- CoreCOM (used for communication between cores)

The CoreCOM shall be used to communicate between the cores. This solution allows the same RTE solution for single core and multicore architectures.

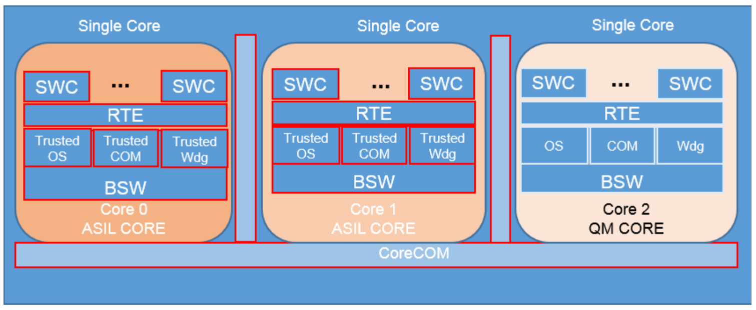


Figure 83: CoreCOM enables communication between cores

The purpose of CoreCOM is to handle the communication between cores (including cores with ASIL requirements) in a multicore system. This is a solution that will be less complex (compared to full OS and RTE implementation as specified by AUTOSAR) and makes it easier to fulfil ASIL requirements compared to implementing multicore communication as specified by AUTOSAR.

Instead of using one RTE for the whole system/ECU, each core will have an RTE of its own. This implies that the ECU extract will need to be split into core extracts.

The CoreCOM will be implemented as a Complex Device Driver (CDD). The CDD will need to know how to provide information from the provider to the consumer and which way it shall be transferred, and this information is included in the multicore extract. From a SWC perspective the interface is unchanged towards the RTE, and no change is required in the RTE. This means that a SWC allocated on one core can be re-allocated to another core in the ordinary "AUTOSAR way". However, the CoreCOM CDD implementation will be architecture dependent (knowing how to connect provider with consumer) and hardware dependent (knowing the most efficient way to communicate between provider and consumer for a specific type of signal. The CoreCOM is generated completely by tools, thus no hand written code is needed.

CoreCOM can realize the communication between the cores in different ways, for example using:

- Shared buffers
- Queued buffers
- Double buffering
- Data-received event over core borders

The different ways of communication have different pros and cons. Which type of communication that is most beneficial for a specific type of communication (size of signal etc.) can be decided on a project basis. For the demonstrator it was decided to use the shared buffer approach.

It is the responsibility of CoreCOM CDD to provide the means of communication between cores for SWCs. The CoreCOM can be generated out of the multicore extract, e.g. for core 1. The main idea of CoreCOM CDD is to hide away the fact that signal sending/receiving and communication between SWCs might traverse core boundaries. SWCs and signals on a core should be entirely oblivious to the fact that the data they send/receive is going to/coming from SWCs or signals from another core. This is done by generating a CoreCOM CDD on each core

From the perspective of an SWC it would look like all of the other SWCs or signals to which it connects are located on the same core. But instead of connecting directly to them it will connect to a CoreCOM CDD port which "emulates" their behavior. Internally, the CoreCOM will take care of all the data transfers by using shared memory. This part is completely hidden from the SWCs and signals in the system.

10.2.6.5.3 Tool support

In order to create an RTE extract that works with the CoreCOM CDD and can be used across multiple cores a tool is required. The tool is responsible for performing this split and is called the MultiCore Extract Generator. The MultiCore Extract contains information about which signals that are communicated to/from the SWC or Service Components allocated on that core (and with which SWC on other cores).

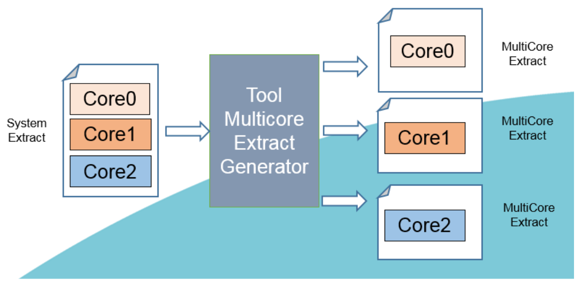


Figure 84: Multicore Extract Generator

10.2.7 Exploitation

The results coming out from the subtask within the Volvo use case will to a large extent be used as input to upcoming research project. The findings and the prototype related to architecture concepts will be handed over to internal projects in which we will continue to study different aspects of a possible future embedded E/E architecture for trucks.

The AUTOMAP tool will be taken over by internal Volvo projects for further validation and for TRL level increase with the target to use it in our product development.

The findings from our SoA activities forms a knowledge base for a national project proposal related to SoA and System-of-Systems.

Arccore has with the research done in EMC^2 been able to strenghten it´s offerings in both the Active Safety domain and functional safety domain. An Autosar platform concept that supports higher ASILs in the Automotive functional safety domain has been created. This is planned to be introduced in the Arccore Autosar Stack together with the development standards of ISO26262 to create a competative Autosar safety offering.

The knowledge of how to create functional safety applications has also been built up together with other partners in the automotive use case. The Arccore Autosar knowledge has been shared with the partners within the use case.

With the knowledge gained in EMC^2 Arccore has significantly raised it´s knowledge in regards to hypervisors, which we believe is a technology that will find it´s way down to the automotive ECU´s.