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	**D6.13 Final definition of the runtime certificate approach incorporating solutions with respect to adaptive behavior**	
Work package	WP6	System qualification and certification
Task / Use Case	T6.3	Certification and Qualification challenges of EMC²-Systems
Lead contractor	Infineon Technologies AG Dr. Werner Weber, werner.weber@infineon.com	
Deliverable responsible	Fraunhofer IESE, Kaiserslautern Dr. Daniel Schneider, daniel.schneider@iese.fraunhofer.de	
Version number	V1.0	
Date	30/09/2016	
Status	Final	
Dissemination level	Public (PU)	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
02A	AVL	Georg Macher	2.7
01O	IESE	Daniel Schneider	1, 2.2.2
01O	IESE	Tiago Amorim	all
01O	IESE	Christoph Dropmann	2.2.1
15E	Tecnalia	Alejandra Ruiz	1.3, 2.4, 2.6, Annex A
15E	Tecnalia	Jason Mansell	1.3, 2.4, 2.6, Annex A
15E	Tecnalia	Angel López	1.3, 2.4, 2.6, Annex A
05A	DANFOSS	Juhua Kusela	2.7

Document History

Version	Date	Author name	Reason
v0.1	21/06/2016	Tiago Amorim	Set-up document
V0.9	20/09/16	Tiago Amorim	Final version before review
V1.0	29/09/2016	Tiago Amorim	Final version after review
	30/09/2016	Alfred Hoess	Final editing and formatting, deliverable submission

Publishable Executive Summary

The purpose of this document is to provide the final definition of the runtime certificate approach, main outcome of the Task 6.3 of the EMC² project. This is the last of a set of three documents, created in the last 30 months of project duration. It is also the only deliverable that is public available. It summarizes the conceptual framework where the approach stands as well as further enhancement, fruit of the work developed in the last 12 months.

Table of contents

List of figures

1. Introduction

Modularization is a trend that is slowly being adopted by current safety standards. Take, for example, the ISO 26262, which is the domain specific functional safety standard for road vehicles. This standard has great developments towards modularity by introducing the Safety Element out of Context [1]. However, this is not enough to achieve the desired flexibility of new systems. This new class of system, the so-called Cyber-Physical Systems (CPS), are software intensive safety critical systems. They could be composed by other systems that have been built at different points in time, by different manufacturers, collaborating with each other at a very low coupling level.

By the middle of 2016, safety standards still require the whole system to be known and defined before certification process can initiate. Once the certification is issued, no modification on the system is allowed. Any change would invalidate the certification and a whole new recertification on the newer version of the system would be required. Some systems are designed to be integrated with others and although we can foresee some usage scenarios and some system candidates the systems to be created will perform functionalities not perceivable at the present time. As complexity grows, awareness of the whole system becomes an unattainable task.

The architecture of systems is changing due to multicore possessors, trend driven by Manufacturers pushing multicore technologies to the market, thus increasing the density of functionalities per ECU (Engine Control Unit). This raises other concerns, such as applications with different criticality running in the same ECU and resource sharing issues.

To tackle those issues, we propose conditional safety contracts designed to be resolved without human interference. The contracts are to be defined in the granularity of applications and platforms where these applications are running. The contracts will describe the safety quality attributes of the composing parts of a system of systems. The assessment of the contracts compatibility is performed at deployment time and runtime by an application-manager without human interference.

1.1 Objective of this document

The objective of this document is to provide the final definition of the EMC² runtime certification approach. This document also summarizes the content of previous two documents (D6.3 and D6.6), that together with the current one, describe the incremental development of the presented approach. The aforementioned documents are confidential while the current document is public.

In the context of the EMC² innovation cycle scheme, this deliverable contributes to the MS7 of technology subprojects (cf. Figure 1).

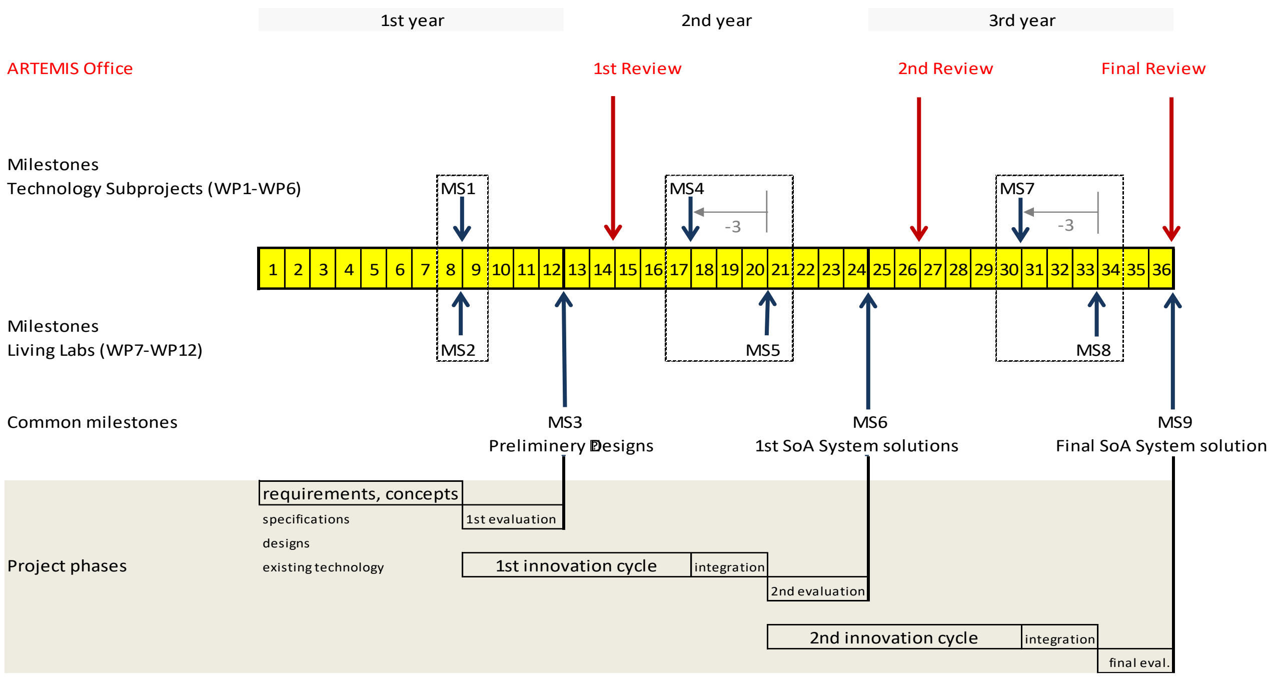


Figure 1 - EMC² milestones plan

1.2 Related deliverables

Note that the following EMC² deliverables are related to this report:

- *D6.6 Refined definition of the runtime certificate approach incorporating first concepts regarding the treatment of adaptive behavior*: This deliverable is the predecessor of this report and the second of a series of three deliverables.

- *D6.8 Final Requirement Set for WP6 – System Qualification and Certification*: This deliverable guides the overall research direction of WP6 and consequently had influence on the development direction of the presented approach.

- *D7.8 Detailed UC T7.3 design, first prototype and preliminary evaluation result*: This deliverable describes the application of the approach in the use case T7.3 "*Design and validation of next generation hybrid powertrain / E-Drive*" in the context of EMC². The runtime certificate approach was implemented in the context of this use case. The experience produced knowledge that served as input to evolve the approach.

- *D10.3 Description of the conceptual architecture:* This deliverable describes the conceptual architecture of use case *T10.1 "Drives and Electric Motors in Industrial Applications"*. The runtime certificate approach was implemented in the context of this use case. The experience produced knowledge that served as input to further develop the approach.

1.3 Relation to other research projects

Our work in EMC² was combined and extended with results from earlier research projects. In this subchapter, we point out how previous research projects influenced our current work.

SafeCer

SafeCer stands for Safety Certification of Software-Intensive Systems with Reusable Components [2]. The project goal was to target increased efficiency and reduced time-to-market by fostering compositional safety certification of safety-relevant embedded systems. The project participants tried to achieve that by providing support for system safety arguments based on arguments and properties of system components as well as to provide support for generation of corresponding evidence in a similar compositional way.

The meeting points of this project and of EMC² are the modular and composability nature of the safety certification developed in both projects. The results of SafeCer were created to be used and applied during development time, more specifically at integration time of the software lifecycle. The approach described here was created with the intention of being put into use during deployment time and runtime (cf. Chapter 0). However, the contracts should be defined at development time. The domains covered by SafeCer were automotive, construction equipment, avionics, and railway.

EVITA

E-safety vehicle intrusion protected applications (EVITA) [3] was a research project whose objective was to design, verify, and prototype an architecture for automotive on-board networks. These networks were connected to security-relevant components that should be protected against tampering. Another requirement was having sensitive data protected against compromise when transferred inside a vehicle.

Future automotive safety applications such as local danger warnings and electronic emergency brakes require security requirements to prevent attacks. Security threats such as forced malfunctioning of safety-critical components or the interference with the traffic flow by means of fake messages are among the problems tackled by this project whose main goal was to achieve secure and trustworthy intra-vehicular communication. EVITA defines the overall functionality of three different hardware security module approaches: –full, medium and light. Moreover, it specifies an elaborate set of functions and their parameters for managing security keys as well as encryption and decryption operations.

A significant number of companies have been active in the EVITA project, including BMW, Continental, Fujitsu, Infineon and Bosch. The results of this project were very useful to define the security properties of the vertical interfaces cf. Subchapter 2.5.1.

PRESERVE

The goal of PRESERVE [4] (Preparing Secure Vehicle-to-X Communication Systems) was to bring secure and privacy-protected V2X (Vehicle-to-Vehicle and Vehicle-to-Infrastructure) communication closer to reality by providing and field testing a security and privacy subsystem for V2X systems. It aims at providing comprehensive protection ranging from the vehicle sensors, through the on-board network and V2V (Vehicle-to-Vehicle)/V2I (Vehicle-to-Infrastructure) communication, to the receiving application. Important subgoals of this project to our work in EMC² were threefold.

- Creation of an integrated V2X Security Architecture (VSA).
- Solving open deployment and technical issues hindering standardization and product pre-development.
- Contribution to on-going harmonization and standardization efforts at the European level.

PRESERVE has emerged from the cooperating entities involved in EVITA.

DECOS

The DECOS (Dependable Embedded Components and Systems) project was a European Integrated Project. The main objective of the project was to make a significant contribution to the safety of dependable embedded systems by facilitating the systematic design and deployment of integrated systems [5]. In order to reach this objective, a generic safety case approach for incremental certification (cf. Figure 2) was developed, which improves the efficiency of the certification process and thus shall facilitate significant cost savings during the development of safety-critical systems.

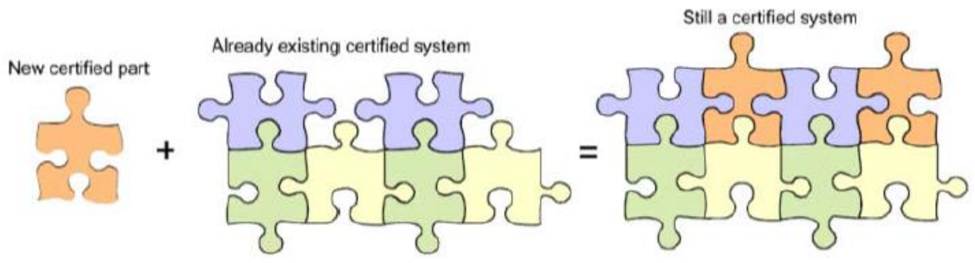


Figure 2 - Incremental certification as envisioned by DECOS [6]

According to [5] and [6], modularity is achieved by separating the certification of core services and architectural services from applications (enabling generic application safety cases (for classes of applications) and for individual (specific) safety cases, supporting independent safety arguments for different distributed application subsystems):

Separating certification of architectural services from certification of applications: The interfaces between the platform and the applications provided via the platform interface are a prerequisite for the separation of the certification of architectural services from the certification of applications.

Separating certification of different distributed application subsystems: The integrated architecture allows the independent certification of different application subsystems, as opposed to considering the system as an indivisible whole in the certification process. The safety argument for each subsystem is provided to the integrator by the suppliers along with the compiled application code of the corresponding subsystem. In order to construct the safety argument for the overall system, the system integrator combines the safety arguments of the independently developed subsystems and acquires additional evidence, such as the results of a formal verification of the architectural services. The decomposition of the overall system into encapsulated subsystems with different criticality levels reduces the overall certification effort and allows focusing on the most critical parts. Furthermore, the separate certification of subsystems is beneficial if functionality is reused in different systems. In this case, the safety argument for the functionality needs to be constructed only once.

DECOS supports both vertical and horizontal modularization (cf. Chapter 2.2). And even though the main focus was on process support, there is also some guidance with respect to the specification of concrete products. Openness and adaptability as characteristics of Cyber Physical Systems (CPS) are, however, not explicitly supported.

Opencoss

OPENCOSS (Open Platform for EvolutioNary Certification Of Safety-critical Systems) [7] was a large-scale collaborative project of the EU's Seventh Framework Program. OPENCOSS focuses on the harmonization of safety assurance and certification management activities for the development of cyber-physical systems in automotive, railway and aerospace industries. The main goal was to reduce both the time and costs overheads inherent to the safety (re)certification of safety critical systems, via a compositional and evolutionary certification approach with the capability to reuse safety arguments, safety evidences and contextual information about system components.

OPENCOSS results include a conceptual framework and the tool platform specified and developed during the project. The conceptual framework mainly corresponds to the CCL (Common Certification Language) and the compositional certification conceptual framework. Figure 3 shows a general view of the functional decomposition for this OPENCOSS tool platform.

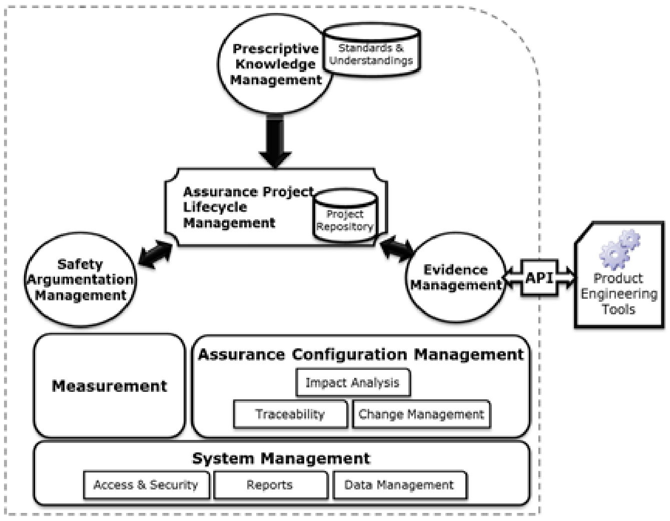

Prescriptive Knowledge Management
Standards & Understandings
Assurance Project Lifecycle Management
Project Repository
Safety Argumentation Management
Evidence Management
API
Product Engineering Tools
Measurement
Assurance Configuration Management
Impact Analysis
Traceability
Change Management
System Management
Access & Security
Reports
Data Management

Figure 3 - Functional decomposition for the OPENCOSS platform

This framework contains the following functionalities:

- Prescriptive Knowledge Management: Functionality related to the management of standards information as well as any other information derived from them, such as interpretations about intents, mapping between standards, etc. This functionality maintains a knowledge database about “standards & understandings”.
- Assurance Project Lifecycle Management: This functionality factorizes aspects such as the creation of safety assurance projects. This function manages a “project repository”, which can be accessed by the other modules.
- Safety Argumentation Management: This functionality manages argumentation information in a modular fashion. It also includes mechanisms to support compositional safety assurance, and assurance patterns management.
- Evidence Management: This functionality manages the full life-cycle of evidences and evidence chains. This includes evidence traceability management and impact analysis. In addition, it is in charge of communicating with external engineering tools (requirements management, implementation, V&V, etc.)
- Assurance Configuration Management: This is an infrastructure function. This includes functionality for traceability management, change management, impact analysis.
- System Management: It includes generic functionality for security, permissions, reports, etc.
- Measurement: This functionality is related to safety indicators, such as Mean Time Between Failures (MTBF).

OPENCOSS results have been benchmarked with the aid of three case studies: (1) an ePARK system for an electric vehicle in the automotive domain, (2) reuse of a railway execution platform in the avionics domain, and (3) the certification of a signaling system in the railway domain.

2. Final definition of the runtime certificate approach

This chapter describes the final definition of the runtime certificate approach, developed during the first 2 years of the EMC² project.

2.1 Concept

The proposed approach complies with the idea that the assurance of the whole composition is necessary. However, instead of having the evaluation completely done at development time, the final parts of the evaluation are shifted to deployment time and runtime, more precisely at the moment of the integration of the systems.

The components (i.e. applications and platforms) and the systems shall assemble the safety critical information of the constituents' parts, this information is then used to certify the overall constellation at deployment and runtime. To achieve this, the constituent parts should be previously certified at development time, and, after knowing the current composition of systems of systems, evaluate the interaction of the building blocks through their characteristics described in contracts.

These contracts are called Multidirectional Conditional Safety Certificates (ConSerts M) [8]. They are multidirectional because they cover both horizontal interfaces (between applications) and vertical interfaces (between platform and application) as well as their specifics (cf. Figure 4). They are modular because of their smallest unit and composability characteristics. They are conditional because the contracts can have open ends (demands) that need to be fulfilled by the environment.

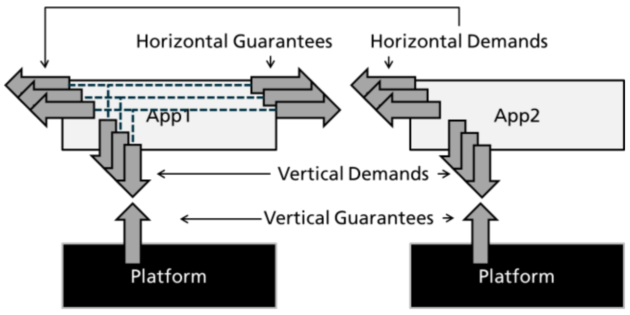


Figure 4 - Vertical and Horizontal Interfaces

The contracts are bound to services that are provided between the components of a system. The contracts are composable which means that a system built up of other systems will have its contract being supported by the contracts of its own subsystems. In the next paragraphs we will introduce the concept of vertical and horizontal interfaces

2.2 Multidirectional interfaces

The concept of vertical and horizontal interfaces was first introduced by Zimmer [9]. The authors distinguish between two types of interfaces: the vertical interface, between the application and the underlying platform, and the horizontal interface, between applications (regardless of whether they are running on the same platform or not) (cf. Figure 4).

The vertical interfaces describe the safety-relevant relations between an application and a platform service. A platform consists of components that provide function-independent services. It enables the hardware that runs the application and the required software to execute the applications independently of the hardware and according to the application's requirements. Platform services are typically developed for reuse, e.g., libraries, communication protocols, or operating systems. Platform developers do not know all future systems that a platform service will be a part of and they do not know in which way the service will be part of the application functionality. Therefore, it is impossible to perform a hazard and risk analysis for a standalone platform service.

To overcome this challenge, we propose a vertical interface description according to [10] following a modular, contract-based approach for the specification of demands and guarantees to form a vertical safety interface. The demands describe the safety-related behavior of the platform as required for the safe execution of the application. Consequently, a demand is linked to a specific application. The guarantees, on the other hand, are linked to a specific platform and define the actual safety-related capabilities of the platform. Mitigation and arbitration are needed to realize safe use of the vertical interface and assure compatibility between application demands and platform guarantees.

On the technical level, the vertical interface is not clearly separated into software and hardware relations. However, the overall viewpoint of our work is the software point of view. Implying that we consider the hardware as a resource used by the software, we do not focus on hardware-specific aspects such as manufacturing technology or special-purpose hardware components. Examples of platforms from the software point of view are AUTOSAR [11] and the ARINC 653 [12] Integrated Modular Avionics standard. Applications are software components that provide system-level functions to end users or other applications to compose more complex functions and are not related to services provided by the platform.

The horizontal interfaces describe relevant (e.g., safety-relevant) relations between applications that enable emergent functionalities that applications would not be able to perform on their own, such as Platooning of Autonomous Vehicles [13] and Tractor Implement Automation [14]. In horizontal relations, there exist the roles of service consumer (which establishes demands to be fulfilled by the consuming services) and service provider (which states guarantees for the provided services). An application can play both roles, being the consumer of some application services while being the provider of services for others. The horizontal interfaces considered in this document describe the safety-relevant relations between applications. These relations are derived from a hazard and risk analysis applied to a top-level applications functionality. The guarantees and demands between applications are associated with services required and provided by applications. However, typically a composition of different applications (and thus systems/devices) join together to render higher level services that could not be rendered by a single system alone. Conceptually, dynamic composition hierarchies are spanned up in this process.

To overcome this challenge of determining safety in a hierarchical, composed service oriented system, we propose a horizontal interface description according to [15] and combine the description with the vertical description to a holistic approach. In summary, if the guarantees are fulfilled by the demands, the applications can function in the way they were designed; otherwise, the available system safety level will be below the specified level.

2.2.1 Vertical Dependencies

In this subchapter, we describe VerSaI, our approach for dealing with vertical safety dependencies. Zimmer [10] proposed a classification of safety-related demand-guarantee dependencies. The dependency classes are platform service, health monitoring, service diversity, resource protection and security services:

- **Platform services** focus on the detection or avoidance of platform failure, e.g., a value failure of a service signal larger than a specified threshold must be detected within a defined period of time.
- **Health monitoring** Application and execution failures get trapped and encapsulated by health monitoring. As an example, the platform has to detect and arbitrate an execution time overrun.
- **Service diversity**, also called dissimilarity or independence, aims at reducing the likelihood of common-cause systematic failures in redundant components. Service diversity focuses on the independence of input services, communication links, and output services.

- **Resource protection** focuses on protection from interferences. We define interference as a cascading failure via a shared resource that potentially violates safety requirements. The interference propagates between several software components via a commonly used resource instead of a private resource for every software component.
- **Security Services** represent services that enrich security properties of the systems. These services are used by security functions and can provide, for instance, encryption services or trusted communication service. This class was developed during the EMC2.

The demands and guarantees are applied for a given confidence attribute. The attribute states the integrity achieved by a guaranty or requested by a demand. Examples are the automotive safety integrity levels (ASILs) or the design assurance levels (DALs) in avionics. The application developer has to specify all the demands that are needed to guaranty safety from the application point of view and has to ensure that the platform service developer has specified guarantees that can be given by the platform. Both guarantees and demands are described using a formal language that is an advancement of the contracted based semi-formal language VerSaI [10]. Chapter 2.4.1 discusses the grammar definition of the language.

The formal language allows automated evaluation of the compatibility of demands with guarantees. As mentioned above, if an application demand can be satisfied by a service guarantee depends on the consumed service guarantees from other applications. Hence, besides the evaluation, if a platform is capable to host a safety critical application, an engineer has to deploy an application into the platform. The current language for the vertical dependencies would support this process too. However, the language focuses on the safety-related dependencies. E.g., it can be specified demands and guarantees with respect to scheduling deadline failure. For the deployment, the information about the deadlines and execution frequency is relevant too. The information is needed to decide if the application can have sufficient CPU time to be executed. The state of the practice is that an engineer, so called integrator, does the deployment of applications into a platform. The current Automotive open system architecture (AUTOSAR) for instance allows only static configuration [16]. Deploying an untrusted application automatically or even during runtime can interfere with safety-critical software.

To this end, we present in [17] an application-download manager concept for automated deploying of applications to an embedded system containing safety-critical software. The download-manager is a software component that automatically evaluates whether the application demands and platform guarantees are compatible and allocates available system resources to applications. From the technical perspective, the application-download manager builds the basis for downloading, installing, and executing applications. From the vertical dependencies, we distinguish between three high-level resource-sharing strategies, which differ in complexity and efficiency: “Don’t Share”, “Don’t Share at the Same Time” and “Regulated Dynamic Sharing”.

The basic idea of the “Don’t Share” strategy is to avoid conflicts between different resource users by allocating private resource. Following this strategy, it is not intended that users access the private resources belonging to others at all. The strategy is rigorous and the decision about the deployment of an application depends only on the fact that the resource (e.g. a CPU) is unused up to now or not.

The basic idea of the “Don’t Share at the Same Time” strategy is to avoid temporal access conflicts between multiple resource users by statically specifying a fixed resource access schedule. Following this strategy, dynamic resource arbitration is avoided and as a consequence, resource access delays are avoided as well. Comparable to the private allocation of resources in the previous strategy, this strategy is more complex because violations by untrusted resource users have to be precluded. E.g. access to the CPU via a faulty application. However, the strategy allows a more efficient sharing of a resource.

The “Regulated Dynamic Sharing” strategy allows dynamic resource access arbitration during run-time. There are different ways to arbitrate concurrent accesses to a resource, e.g. priority-based, FIFO-based or round robin arbitration. To decide if an application is able to use the resource and thus can be deployed, the resource access behavior needs to be specified. This is done in terms of upper and lower limits for resource usage, which often includes the duration and frequency of resource access. Compared with the previous strategies, dynamic sharing offers more flexibility. However, it is more complex and we cannot assume that every resource user in a mixed-critical system adheres to these predefined limits. Therefore, a protective entity that monitors resource access during runtime is needed.

To enable an automated deployment, the platform developer has to realize a sharing strategy for every platform service that is provided to applications. In addition, safety mechanisms need to ensure at runtime that applications do not consume more platform resources than demanded by an application.

Within our application-download manager, the resource protection between mixed critical applications (e.g. non-safety-critical application and safety-critical software running on the platform) is achieved via three sets of protective measures: a) measures against spatial memory corruptions, b) measures against temporal scheduling interferences and c) measures against corruptions via shared platform services.

In summary, the vertical language of our method allows to specify safety related dependencies between applications and platforms along the five presented classes of vertical dependencies. In addition, the language allows, in combination with resource sharing concepts, as realized within the presented application-download manager, an automated vertical deployment.

2.2.2 **Horizontal Dependencies**

The horizontal interfaces are addressed in ConSerts [18] [19] [20]. ConSerts stands for Conditional Safety Certificates. The approach utilizes modular conditional certificates and operates between horizontal interfaces of systems. ConSerts are post-certification artifacts (i.e., certification has been conducted in the traditional way) equipped with variations points bound to formalized external dependencies that are meant to be resolved at runtime. This characteristic is what makes the certificates “conditional” and provides the flexibility in the certificates that is required to be useful for a sufficiently wide range of concrete integration scenarios. The conditional certificates must also be modular in order to conduct the certification process at the level of the units composing the targeted systems of systems.

The conditional certificates are to be evaluated automatically and autonomously by the system at the moment of integration at runtime, based on runtime representations of the certificates of the involved compositional units. Once all conditions have been resolved and the evaluation is finished, an overall certificate variant can be determined for the actual composition that has been formed. In a sense, the final certification step has thereby been postponed to runtime and we can thus speak of “runtime certification”.

Whenever the overall system composition changes or the system adapts itself, a re-evaluation of the conditional certificates must be conducted and the overall certificate for the composition must be updated. Such a re-evaluation might well be triggered by a minor dynamic adaptation in one of the subsystems or even an update, which, however, can easily trigger a chain reaction in related components leading to complex reconfiguration sequences. Therefore, there is a strong interdependency between dynamic adaptations and the dynamic evaluations of the conditional certificates. An adaptation might lead to an invalidation of the current certificate and thus to a re-evaluation and the determination of a new certificate. This might then violate given top-level trust requirements, which might again trigger additional adaptations in order to regain sufficient trust guarantees (e.g., via graceful degradation, which could imply a loss of application features).

2.3 ConSerts M Contracts' mechanics

The contracts are to be evaluated automatically by the composing systems before any service consumption and provision. In the horizontal level, each service composing a safety critical functionality, provided by every unit of composition, shall have its own contract. Adaptive systems might need different contracts in different modes of operation (if it affects the functionality). In the vertical level, the platform contains a single contract that describe all vertical guarantees for all services. The vertical demands of an application are related to further horizontal guarantees of services or to services designed for the final user. On the vertical level, the contracts are assessed latest at deployment time. In case of horizontal interfaces, the contract evaluation can be postponed to runtime.

Services can be provided directly to the end user or to another component that uses it to build more complex services. Providing raw sensor signal is a type of service that is consumed by other modules while airbag activation is a service consumed by the end user. There will be cases when the detachment between application and platform does not make sense, having a system like contract (i.e. platform and application are described using a single contract). A gas pedal component, for example, would have such type of contract.

The conditional nature of ConSerts M contracts requires some variants to be fulfilled. Demands are requirements outside the boundaries of a module. They represent safety requirements that cannot be verified at design time. A service guarantee can be bound to the fulfillment of one or more demands. Demands are related to quality attributes of safety properties required to exist in the environment (e.g. horizontal services being consumed or platform guarantees). They are related to sub-services (functionalities of other systems) required to render more complex services (horizontal interfaces) or safety related characteristics of the underlying platform. The demands and guarantees are defined using integrity levels which are domain dependent (i.e. the automotive safety integrity levels (ASILs) or the design assurance levels (DALs) in avionics domain).

In case of addition, removal, replacement or adaptation of modules in an already certified SoS, a re-evaluation (i.e. assessment of the contracts w.r.t. demands and guarantees) must be carried out and the new composition certificate updated. If, among the systems, there are adaptive ones suffering adaptations that change the contracts of its services, a re-evaluation shall also be carried out at individual certificate level.

The re-evaluation starts locally with the new contracts and might propagate throughout the other systems towards the root in a chain reaction fashion. If the new configuration cannot reach the safety requirements of the top level service, a reconfiguration of the system of systems is required to regain sufficient trust guarantees (e.g. by graceful degradation, consequently causing a loss of application features).

ConSerts M contracts intend only to shift parts of the certification process to runtime, thus the contracts for the applications and platforms are still issued by safety experts, independent organizations, or authorized bodies in the same fashion as current safety certifications are issued currently (i.e. claims on the fulfillment of safety requirements proved through proper evidence). Those contracts can be reused and can lose validity if the application (or use context) changes.

A use case: lately, Tesla released the 8th version of its auto pilot to be updated over-the-air on its cars. However, due to application vertical demands, this new functionality will only be available to cars produced after 2014. In a scenario where Tesla would detach itself from the car manufacturing business and had the auto pilot as single product, a consumer could upgrade the processor (or radars, or other applications) of her car to allow the new auto pilot to be installed. These subsystems, from different producers, are assembled together to render a safety critical functionality. The composition should be assessed to assure an acceptable level of safety and a solution for this can be the use of ConSerts M.

2.4 Contract description

We propose the use of a formal contract to indicate information about demands and guarantees of the components which form its interfaces. It uses xml tags as a way to formalize the parameters but also provide a human understandable mean to read the information. The contracts structure is as follows:

```
<M2C2>
      <ComponentName> Name </ComponentName>
      <Guarantee>
      …
      </Guarantee>
</M2C2>
```

Within the `<M2C2>` tag the whole contract is described. Then the following tag is used to indicate the name of the component, owner of the contract. After the definition of the components we shall describe the guarantees. We have two possibilities for this. In case the component is a platform the contract will describe the vertical guarantees being provided. The vertical guarantees are described using the following tags:

```
<Vertical_Guarantee>
…
</Vertical_Guarantee>
```

In case the component is not part of the platform but an application, then we will indicate the horizontal guarantees. These are described using the following tags:

```
<Horizontal_Guarantee>
…
</Horizontal_Guarantee>
```

Within these tags, both `Vertical_Guarantee` and `Horizontal_Guarantee` we shall specify the guarantees description as well as the demands directly dependent on the validity of those guarantees. An application service integrity level is defined based on the fulfilment of the demands considering both Vertical and Horizontal Interfaces.

2.4.1 Vertical Contracts

The vertical interfaces describe technical safety requirements. They are always realized between the platform, which always guarantees the properties, and the application, which is always a demander. We can consider the vertical interfaces as Services provided to application by the platform (including its Hardware Abstraction Layer). With the vertical interfaces specification, we aim to describe the needs/guarantees of resources without compromising the application intended execution. When specifying the vertical interfaces we can classify them into four classes, as mentioned in chapter 2.2.1: Platform Services, Health Monitoring, Service Diversity and Resource Protection. An example of a Platform service specification is described within the following tags:

```
<Platform_Service>
      <Failure>Message Corruption </Failure>
      <Reaction>detected</Reaction>
      <IntegrityLevel>A </IntegrityLevel>
      <Error>1.2ms</Error>
      <Latency>less than 5 ms + - 2 us</Latency>
</Platform_Service>
```

The complete details of the specification are listed in the annex.

2.4.2 Horizontal Contracts

The Horizontal Interfaces happens between applications. A horizontal contract starts describing the guarantee of a service and its associated demands.

```
<Horizontal_Guarantee>
      <ConfigurationName> serviceName </ConfigurationName>
      <IntegrityLevel> ASIL_level </IntegrityLevel>
      <SafetyProperty>
            …
      </SafetyProperty>
      <DemmandSet>
            <Horizontal_Demand> … </Horizontal_Demand>
      </DemmandSet>
</Horizontal_Guarantee>
```

The tag `ConfigurationName` names the service being provided by this guarantee. Then `IntegrityLevel` states the overall integrity level of the service. A safety property refinement comes in sequence with the tag `SafetyProperty` where the properties of the service are characterized. Then we find the `DemandSet` tag, in which the demands necessary for providing the service with the stated shall be defined. If we are including a horizontal interface for a service which shall be provided by another component, then the information shall be surrounded by the tag `<Horizontal_Demand>.` If we specify a service from the platform them the demand shall be surrounded by the tag `<Vertical_Demand>`. Check the annex for the complete specification.

2.5 Security in contracts

Until this point, we have only considered safety properties in the contracts. However, due to the open nature of cyber physical systems, security shall also be taken into consideration if we want to keep systems safe. Security is the capacity of a system to withstand malicious attacks. These are intentional attempts to make the system behave in a way that it is not supposed to. Both safety and security contribute to the system's dependability, yet each in its own way. Therefore, safety can no longer be considered in an isolated and independent way from security. Since CPS are open heterogeneous interconnected systems of different manufacturers, security threats are inevitable [21].

Safety is a constant characteristic of a static system which, ideally, is never changed once deployed. Security requires constant vigilance through updates to fix newly discovered vulnerabilities or improve mechanisms (e.g., strengthening a cryptographic key). Security properties becomes less efficient as time goes due to increases in computational power, development of attack techniques, and detection of vulnerabilities and design flaws. This is such a huge issue that a system might require a security update the day after it goes into production [22].

Security for safety is indispensable in future to enable CPS use. Therefore, the ideas here described are a contribution to support the development of safe and secure CPS. In this subchapter, we present the initial conceptual ideas of how security can be realized in contracts considering both vertical and horizontal perspectives.

2.5.1 Secure Vertical Interfaces

The secure vertical interfaces describe services provided by the platform to applications. These services are used to compose security services or give/enhance security attributes to/of the system. While the safety vertical interfaces are related to the quality of service, the secure vertical interfaces provide functionalities that are designed to be used by the application to render security functions. The security measures of an application can be implemented by software, hardware or both. By using hardware, engineers can improve the performance of cryptographic algorithms, delivering extra performance to the application.

As presented in §1.3, the result of project Evita and PRESERVE was a hardware architecture to address security needs of the applications running in security-critical platforms. The architecture component proposed was called Secure Hardware Extension (SHE). The SHE specification defines a set of functions and a programmer's model (API) that allows a secure zone to coexist within any electronic control unit installed in the vehicle. Currently, manufacturers already produce chips equipped with SHE [23], which are basically a suite of security services provided by the platform such as the storage and management of security keys, plus encapsulating authentication, encryption and decryption algorithms that application code can access through the API. These features help maximize flexibility and minimize costs.

Considering those services, and how they can influence safety characteristics of systems, we created a novel class of platform services to be represented in the ConSerts M contracts regarding the Vertical Interfaces. An application developer can, depending on the applications security needs, specify corresponding application demands. This class is called Security Services and they are represented in the contracts within the vertical demands or guarantees part with the following xml tags:

```
<Vertical_Guarantee>
      <Security_Services>
            <Key_Management>10</Key_Management>
      </Security_Services>
</Vertical_Guarantee>
```

Within the `Security_Services` tag we can define the following security services:

Key management

```
<Key_Management></Key_Management>
```

This service allows to store and retrieve keys from a secure part of the platform. The number of keys that the platform can handle or that an application requires is a variable property of this service. If more than one application running in the same platform requires keys to be handled by the platform, the overall number of keys need to be shared among the applications. Since the number of keys available through the life-time of a system may vary, this information, when being provided by the platform, needs to be processed by the application-download manager to figure out how many keys are available.

Symmetric data encryption/decryption

```
<Symmetric_Encryption>ECB, CBC, OFB, CFB CTR, XTS, GCM</Symmetric_Encryption>
```

This service is provided by platforms equipped with an instruction set for improving the speed of applications performing encryption and decryption using the Advanced Encryption Standard (AES). By using this service, the application performance increases considerably. Table 1 enumerates some well stablished encryption methods and provides the indentifier that should be used to identify such method in the contracts.

Modes of operation	ConSerts M indentifier
Cipher Block Chaining	CBC
Electronic Code Book	ECB
Cipher Feedback	CFB
Output Feedback	OFB
Counter	CTR
Galois/Counter Mode	GCM
XEX-TCB-CTS	XTS

Table 1 - Encryption types and its ConSerts M wording

MAC generation /verification

```
        <MAC_Gen_Ver></MAC_Gen_Ver>
```

An application using this service is able to generate a message authentication code (MAC) and verify the MAC from messages received. This prevents that untrusted equipment sends and receives messages from the applications running in the platform.

Random number management

```
        <Random_Number_Management> PRNG | TRNG </Random_Number_Management>
```

Random numbers are required to generate keys to cryptograph messages. With a fair amount of messages, an attacker might be able to figure out the algorithm being used to generate those random numbers and crack the messages. A platform equipped with such functionality could provide random number with two levels of confidence, a pseudorandom number generator (PRNG) and a true random number generator (TRNG)

Secure boot

```
        <Secure_Boot></Secure_Boot>
```

Secure Boot is a security standard allowing making sure that a system boots using only software that is trusted. When the system starts, the firmware checks the signature of each piece of boot software, including firmware drivers and operating system. If the signatures are good, the system boots, and the firmware gives control to the operating system.

Asymmetric encryption/decryption

```
        <Asymmetric_Encryption></Asymmetric_Encryption>
```

Like the symmetric encryption service previously presented, the platform can provide more efficient asymmetric encryption/decryption services to the application through the use of specific instructions sets. The goal here is also to improve the efficiency of such operations.

Typical applications of the security services are tuning protection, where unauthorized modifications are detected by the system, the car wireless immobilizer, to prevent car theft, secure on-board communication, to prevent hacking or remote controlling. The Security Services do not intend to replace the API specification of the services; they are models used at runtime [24] to describe security properties. Further detail on the functions themselves can be found in the deliverables of EVITA Project [3].

2.5.2 Secure Horizontal Interfaces

The horizontal aspect of the contracts is related to functional characteristics of the systems and this also applies for security properties. Security requirements are a class of Non-Functional Requirements (NFRs) that relate to system confidentiality, integrity and availability. In CPS, they are related to safety-critical services and to mechanism to guarantee the aforementioned properties. This means they will always be described within the service configuration description, whether it is a demand or a guarantee.
Let’s take for example a gas pedal and an ECU connected to a CAN network within a car. The car producer wants to prevent unintended acceleration of the car due to messages being sent through malicious attacks. In the next lines the reader can see an example of Security Property:

```
<Horizontal_Demand>
        <ConfigurationName>setAcceleration</ConfigurationName>
        <IntegrityLevel>QM</IntegrityLevel>
        <SafetyProperty>
                <Commission>B</Commission>
```

```
      </SafetyProperty>
      <SecurityProperty>
            <SymetricEncryption> ECB, CBC, OFB, CFB CTR </SymetricEncryption>
      </SecurityProperty>
</Horizontal_Demand>
```

In this case, the service setAcceleration should also provide support for symmetric encryption to fulfill this demand. A guarantee to this demand would look alike:

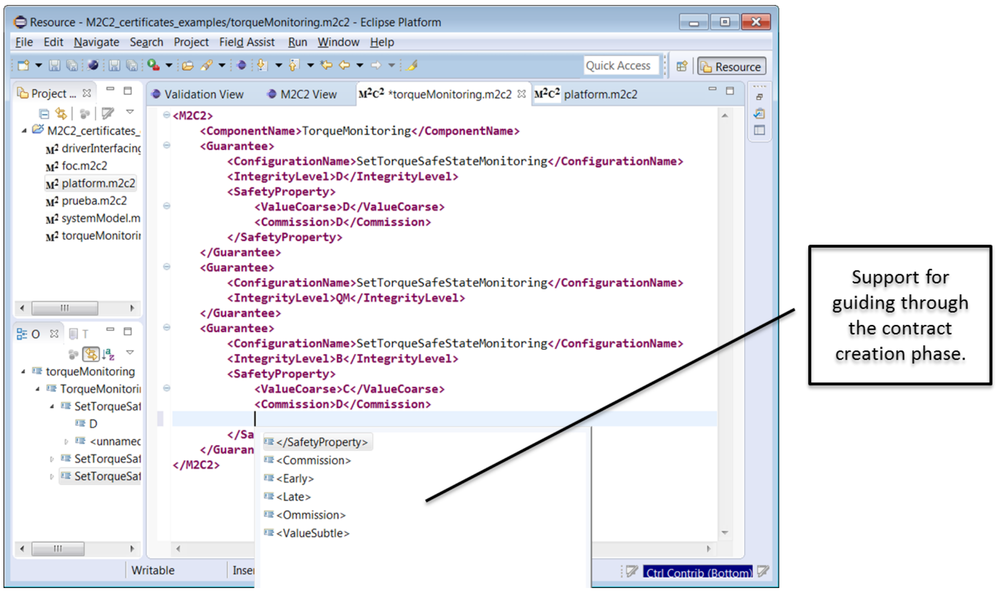
```
<ComponentName>GasPedal</ComponentName>
<Horizontal_Guarantee>
      <ConfigurationName>setAcceleration</ConfigurationName>
      <IntegrityLevel>QM</IntegrityLevel>
      <SafetyProperty>
            <Commission>B</Commission>
      </SafetyProperty>
      <SecurityProperty>
            <SymetricEncryption> ECB, CTR </SymetricEncryption>
      </SecurityProperty>
</Horizontal_Guarantee>
```

Despite the measures taken during development, eventually a security update will be required throughout the life cycle of a product. Also the security mechanisms might become obsolete and need to be replace by more efficient ones. New contracts would be required and after a software update, the contracts compatibility should be assessed. New/modified algorithms not supported by existing hardware could make systems obsolete.

The development of security properties for horizontal contracts is still in its early stages by the time of the compilation of this document. However, the language can be easily extended and further developed, which is the goal of our future work. Besides, the new security measures to be created will eventually require new contract details to be developed.

2.6 Tool for contract creation

A contract creation tool was built in cooperation between FhG IESE and Tecnalia and was presented in the project second year review as demonstrator. The tool is an implementation of the ConSerts M concept and supports the creation of the contracts through auto-complete and syntax checks. The tool includes support for the contracts edition, suggesting to the user the different options valid for the horizontal and vertical dependencies identification for each of the components. The use of the tool has provided a powerful mechanism to ensure the edition of contracts with no errors in the grammar defined and reducing the learning curve for new users of this approach.



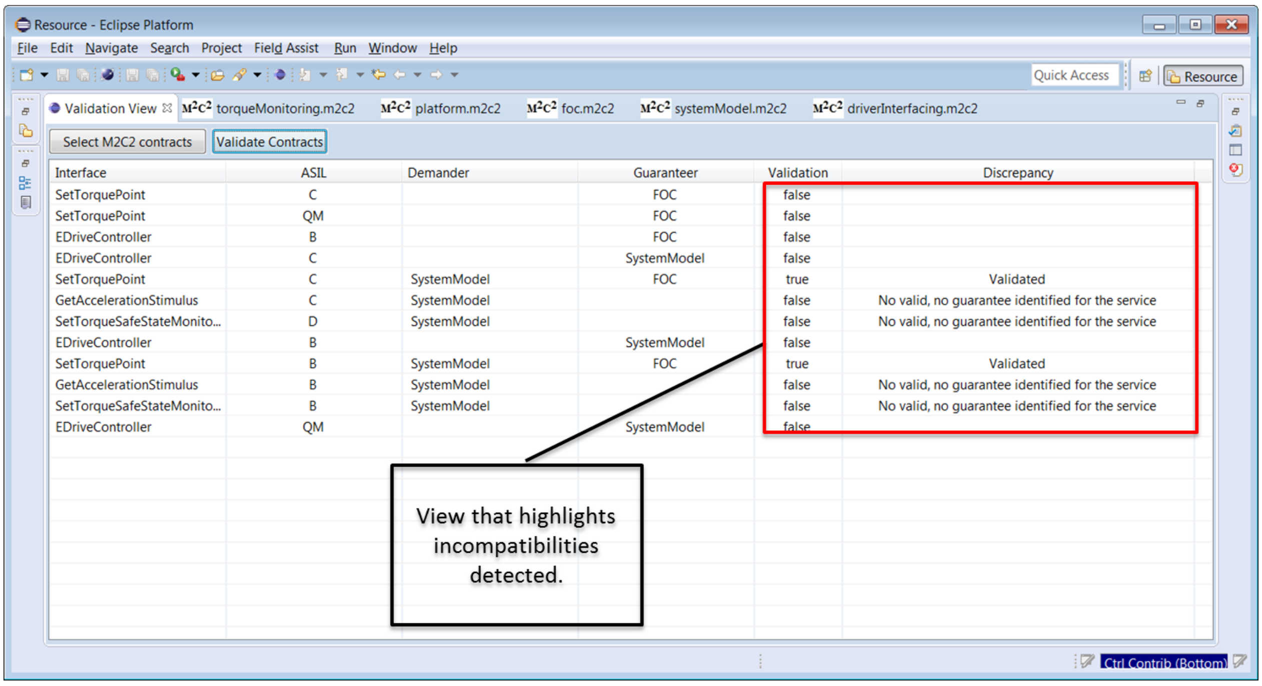
```
<M2C2>
    <ComponentName>TorqueMonitoring</ComponentName>
    <Guarantee>
        <ConfigurationName>SetTorqueSafeStateMonitoring</ConfigurationName>
        <IntegrityLevel>D</IntegrityLevel>
        <SafetyProperty>
            <ValueCoarse>D</ValueCoarse>
            <Commission>D</Commission>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

Resource - M2C2_certificates_examples/torqueMonitoring.m2c2 - Eclipse Platform
File Edit Navigate Search Project Field Assist Run Window Help
Quick Access
Resource
Validation View
M2C2 View
*torqueMonitoring.m2c2
platform.m2c2
</SafetyProperty>
<Commission>
<Early>
<Late>
<Ommission>
<ValueSubtle>
Writable
Ctrl Contrib (Bottom)
Support for guiding through the contract creation phase.

It also allows checking if the contracts demands will be fulfilled by other contracts guarantees at design time. The lack of grammar errors is especially important for a second step, the validation of the demands once we integrate the components. The tool implemented also provides support to check the fulfilment of the demands, either horizontal demands or vertical demands. The verification feature will check whether the demands are being fulfilled by the guarantees, going through all contracts from the system of system constellation, in order to assure enough guarantees before allowing the functionality to be carried on.

When an incompatibility is identified a message is provided to the user indicating the reason for the incompatibility. At this point the user should make a deeper analysis to check. The responsible should verify if the assurance requirements for the system allows an incompatibility in the demand providing that the signal is available, however with a lower ASIL compliance and then it will be up to the integrator responsibility to include external safety measures to ensure the safety standard demands



Interface	ASIL	Demander	Guaranteer	Validation	Discrepancy
SetTorquePoint	C		FOC	false	
SetTorquePoint	QM		FOC	false	
EDriveController	B		FOC	false	
EDriveController	C		SystemModel	false	
SetTorquePoint	C	SystemModel	FOC	true	Validated
GetAccelerationStimulus	C	SystemModel		false	No valid, no guarantee identified for the service
SetTorqueSafeStateMonito...	D	SystemModel		false	No valid, no guarantee identified for the service
EDriveController	B		SystemModel	false	
SetTorquePoint	B	SystemModel	FOC	true	Validated
GetAccelerationStimulus	B	SystemModel		false	No valid, no guarantee identified for the service
SetTorqueSafeStateMonito...	B	SystemModel		false	No valid, no guarantee identified for the service
EDriveController	QM		SystemModel	false	

2.7 Application in EMC² living labs

This chapter describes how the contracts were applied to the living labs. Some of the work described here is ongoing work. The implementation efforts are undertaken until the MS8.

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

This use case is being driven together by AIT, AVL, BUT, SFR, TUW, TNO, VIF, IESE, Tecnalia. It focuses on the challenges related to development of next generation hybrid powertrains. The interdisciplinary knowledge required for hybrid powertrain development (combustion engine, battery storage, transmission, e-drive and many others) and the large number of combinations for energy generation and transfer result in a fast growing complexity of the involved control system E/E architecture.

The second version of the hybrid powertrain use case demonstrator (UC7.3) provides a multi-core platform for the integration of two mixed-critical automotive controller units (Electronic Motor Control Unit and Vehicle Control Unit) in one (EMCU + VCU = VEMCU) control unit.

Within this setup (block diagram can be seen in Figure 5), current hybrid/electrical powertrain applications approaches and functional integration of high dynamic e-drive system controls with time based vehicle control algorithm are tackled.

Nevertheless, with the current integration of external information and connected vehicle features, and the ongoing advance towards automated and autonomous driving, an open system like the hybrid powertrain, must be able to integrate new applications, e.g. torque monitoring, during the product's lifetime. In such a scenario, we assume that the application developer has specified all the demands that are needed to guaranty safety from the application point of view and that the platform service developer has specified guarantees that can be given by the platform. This supports the integration of over the air (OTA) updates of vehicle functions as well as SOA concepts, ensuring the migration of crucial functions to other control units, for dependable automotive functions.

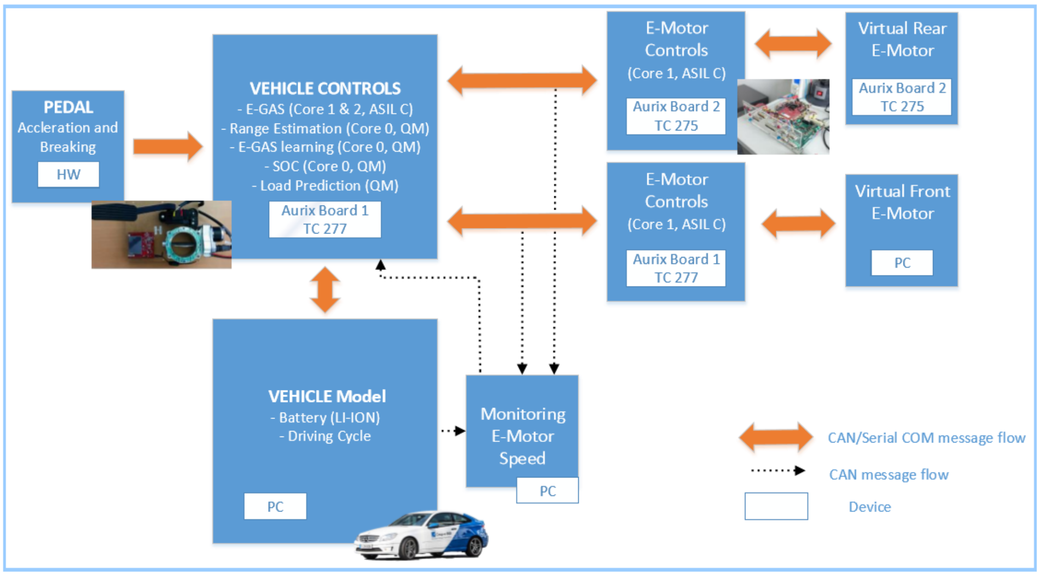


Figure 5 – Demonstrator architecture

T10.1 UC Drives and Electric Motors in Industrial Applications

The T10.1 describes a typical industrial application (cf. Figure 6) automated and controlled using multiple devices. These devices have one or more mechanical parts, which are driven by an electric motor and

Variable Speed Drives (VSD) that operate these motors via a transmission system. VSDs can give precise control over speed, torque, and position, which can then be matched to the process under control, yielding energy savings and removing the need for complicated mechanical control devices.

A VSD follows a standard high-level design, which includes an Embedded Control System that: communicates with external controllers via the External Signals; actuates the Motor via the Power Section, to fulfil the application requirements and requests received via External Signals; and finally, uses current and speed feedbacks from Sensors to the control system.

It is typical that in addition to safety requirements – a characteristic of numerous automation systems – VSDs must satisfy other quality related requirements. For instance, a crane manufacturer must also fulfill performance (e.g. vertical speed), user comfort (e.g. smooth acceleration), customer expectations (e.g. ability to serve 50 meters vertical height), and reliability (e.g. 2 hours maximum maintenance time per year) requirements.

The ConSerts M will be applied to allow the drive function binding to be performed at different times, allowing a certain degree of adaptation to the system. Binding of functionality should take place during the system installation or at runtime.

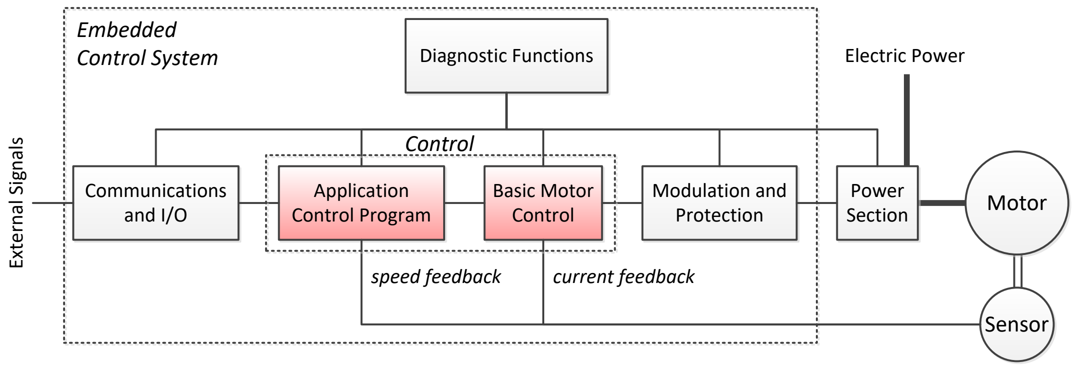


Figure 6 - Functional elements of a VSD

The basic vertical integration needs between various SW components within the drive are declarative. Syntactic compatibility including required and provided ports and their datatypes is checked. Runtime dependences are declared. Our tool has been inspired by the work done by FhG IESE and Tecnalia. Changes include management of execution order and frequency. We also use ended up using JASON instead of XML. These internal interfaces are not certified.

Interfaces between large components are signed and encrypted where necessary:

- Application software package or individual application
- Firmware and firmware interface
- Extension devices: I/O boards, fieldbus options, wireless gateways

These interfaces are seen as secure against counterfeit. Checks are made during configuration time.

In the industry automation systems VSD is seen as a component. Horizontal interfaces are towards other drives and controlling systems. In these interfaces, we are dependent on industry standardization (fieldbuses). We are planning to support secure protocols on all interfaces. Some of these interfaces can carry safety related information. These are handled so that requirements set by the desired SIL level can be fulfilled.

3. Conclusion and Future work

In summary, ConSerts M is a solution concept to guarantee safety for open and adaptive multi-core systems. The main idea is to split the safety relevant dependencies within a system, or system of systems, at smallest granularity (i.e. applications and platform) yet keeping scalability. Demand and guarantee based conditional certificates are used as means of specification. At runtime, the ConSerts M framework autonomously evaluates if the safety relevant interfaces are valid and compatible and what top-level guarantees can be offered for a given composition. In addition, we present a first idea of security extension of ConSerts M to consider security dependencies via vertical and horizontal certificates too.

The feasibility and limitations of ConSerts M were explored in small case studies in the past. Current work is to apply ConSerts M in the living labs and combine the ConSerts M concept for downloading applications to an embedded system containing safety-critical software. As future work, we would like to develop further and validate in a case study the security-dependencies description.

4. References

[1] International Organization for Standardization, "ISO/DIS 26262 Road vehicles — Functional safety," International Organization for Standardization, 2011.

[2] "SafeCer," [Online]. Available: http://www.safecer.eu/. [Accessed 03 08 2016].

[3] "EVITA," [Online]. Available: http://www.evita-project.org/. [Accessed 03 08 2016].

[4] "PRESERVE Project," 2015. [Online]. Available: https://www.preserve-project.eu/. [Accessed 16 09 2016].

[5] H. Kopetz, R. Obermaisser, P. Peti and N. Suri, "From a Federated to an Integrated Architecture for Dependable Embedded Real-Time Systems," TU Vienna University of Technology, Austria, and Darmstadt University of Technology, 2004.

[6] E. Althammer, E. Schoitsch, G. Sonneck, H. Eriksson and J. Vinter, "Modular certification support — the DECOS concept of generic safety cases," in 6th IEEE International Conference on Industrial Informatics (INDIN), 2008.

[7] "Open Platform for EvolutioNary Certification of Safety-critical Systems," [Online]. Available: http://www.opencoss-project.eu/. [Accessed 19 09 2014].

[8] T. Amorim, A. Ruiz, C. Dropmann and D. Schneider, "Multidirectional Modular Conditional Safety Certificates," in 4th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems - SAFECOMP, Delft, 2015.

[9] B. Zimmer, S. Bürklen, M. I. Knoop, J. Höfflinger and M. Trapp, "Vertical Safety Interfaces - Improving the Efficiency of Modular Certification," in SAFECOMP 2011, 2011.

[10] B. Zimmer, "Efficiently Deploying Safety-Critical Applications onto Open Integrated Architectures," Fraunhofer IESE, Kaiserslautern, 2014.

[11] AUTOSAR, "Website of the autosar standard," [Online]. Available: http://www.autosar.org/. [Accessed 27 07 2015].

[12] ARINC, "Arinc 653, avionic application software standard interface, part 1," 2005.

[13] P. Fernandes and U. Nunes, "Platooning of Autonomous Vehicles with Intervehicle Communications in SUMO Traffic Simulator," in International IEEE Conference on Intelligent Transportation Systems (ITSC), 2010.

[14] M. Hoyningen-Huene and M. Baldinger, "Tractor-Implement-Automation and its application to a tractor-loader wagon combination," in 2nd International Conference on Machine Control & Guidance, University of Bonn, Germany, 2010.

[15] D. Schneider and M. Trapp, "Engineering Conditional Safety Certificates for Open Adaptive Systems," in In Proc. of the 4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS), pp. 139-144, York, 2013.

[16] M. e. a. Wagner, "Towards runtime adaptation in AUTOSAR: Adding," in ETFA, 2014.

[17] C. e. a. Dropmann, "An application software download concept for safetycritical," in CARS, EDCC, 2015.

[18] D. Schneider and M. Trapp, "Conditional Safety Certificates in Open Systems," in Proceedings of the 1st Workshop on Critical Automotive Applications: Robustness & Safety (CARS'10), 2010.

[19] D. Schneider and M. Trapp, "A Safety Engineering Framework for Open Adaptive Systems," in In Proc. of the Fifth IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2011.

[20] D. Schneider and M. Trapp, "Conditional Safety Certification of Open Adaptive Systems," ACM Trans. Auton. Adapt. Syst. 8, p. Article 8, 2013.

[21] D. Schneider, E. Armengaud, E. Schoitsch and , "Towards Trust Assurance and Certification in CyberPhysical Systems," in SAFECOMP Workshop, Firenze, 2014.

[22] T. Amorim, D. Schneider, V. Y. Nguyen, C. Schmittner and E. Schoitsch, "Five Major Reasons Why Safety and Security Haven't Married (Yet)," ERCIM News, pp. 16-17, 07 2015.

[23] Infineon, "Highly integrated and performance optimized 32-bit microcontrollers for automotive and industrial applications," [Online]. Available: http://bit.ly/2czXi4I. [Accessed 18 09 2016].

[24] M. Trapp und D. Schneider, „Safety assurance of open adaptive systems - a survey,“ Lecture Notes in Computer Science, Bd. 8378, Nr. Models@Run.Time: Foundations, Applications, and Roadmaps, p. 279–318, 2014.

ANNEX A: ConSerts M grammar specification

In this chapter we present the ConSerts M grammar specification using BNF notation. In the BNF specification, we use the conventions: {def}* to denote 0 or more repetitions def, and {def}+ to denote 1 or more repetitions of def. Also, by names, we assume infinite sets of names, as follows:

- Blocks' names represented in bold.
- Also, n,m will serve as typical representatives for the Natural numbers, d as double number

```
<M2C2>
        <ComponentName> ComponentName </ComponentName>
        (HorizontalGuarantee | Vertical_Guarantee)*
</M2C2>
```

`M2C2:` indicates the start and end of the contract
ComponentName: a text including A-Z, a-z, 0-9. The name cannot start with a number.

HorizontalGuarantee:

```
<Horizontal_Guarantee>
        <ConfigurationName> ConfigurationName </ConfigurationName>'
        <IntegrityLevel> IntegrityLevel </IntegrityLevel>'
        (<SafetyProperty> Property+=SafetyProperty
        </SafetyProperty>)*
        (ANDDemandSet | ORDemandSet | DemandSet)*
</Horizontal_Guarantee>
```

`Horizontal_Guarantee:` Denotes that the specification between these tags refers to an horizontal interface which is guarantee to other components which demands it.
ConfigurationName: a unique name of the service, is specified by text including A-Z, a-z, 0-9. The name cannot start with a number
IntegrityLevel: could be any of these values: `'A','B','C','D','QM'`
`SafetyProperty:` indicates that the specification between these tags refers to a properties characteristic associated with the service indicated in the *ConfigurationName*

Property:

```
        (<ValueCoarse> IntegrityLevel </ValueCoarse>)*
        (<Commission> IntegrityLevel </Commission>)*
        (<Ommission> IntegrityLevel </Ommission>)*
        (<ValueSubtle> IntegrityLevel </ValueSubtle>)*
        (<Early> TimeProperty </Early>)*
        (<Late> TimeProperty </Late>)*
```

`ValueCoarse`*:* the integrity level provided in case the value is deviated from the set value by more than 20%.
`Commission`: the integrity level provided in case the value is provided even though it has never been requested.
`Omission` the integrity level provided in case the value is never provided even though it has been requested.
`ValueSubtle:` the integrity level provided for the precision of the service provided.
`Early:` between these tags we shall include information about the time. It indicates the maximum time in advance a service can be provided
`Late:` between these tags we shall include information about the time. It indicates the maximum delay time a service can be provided

ANDDemandSet:

```
        <AND>
                DemandSet
        </AND>
```

ORDemandSet:

```
        <OR>
        DemandSet
    </OR>
```

DemandSet:
```
    <DemmandSet>
        (Demand)*
    </DemmandSet>
```
;
Demand:

HorizontalDemand | **Vertical_Demand**

HorizontalDemand:
```
<Horizontal_Demand>
    <ConfigurationName> ConfigurationName </ConfigurationName>
    <IntegrityLevel> IntegrityLevel </IntegrityLevel>'
    (<SafetyProperty> Property+=SafetyProperty
    </SafetyProperty>)*
</Horizontal_Demand>
```

It has the same structure as a horizontal guarantee. See **HorizontalGuarantee** block for details.

TimeProperty:
```
<IntegrityLevel> IntegrityLevel </IntegrityLevel>
<SPSpec> n time_unit </SPSpec>
<Situation>' situation </Situation>
```

time_unit: It could be either 'ms' or 'us'
situation: a text indicating the context environment

Vertical_Guarantee:
```
<Vertical_Guarantee>
    (PlatformService | ServiceDiversity | HealthMonitoring | ResourceProtection)
</Vertical_Guarantee>
```

`Vertical_Guarantee:` Between the tags, we shall specify the guarantee about the services provided by the platform.

Vertical_Demand:
```
<Vertical_Demand>
    (PlatformService | ServiceDiversity | HealthMonitoring | ResourceProtection)
</Vertical_Demand>
```
It has the same structure as a vertical guarantee. See **VerticalGuarantee** block for details.

PlatformService:
```
    <Platform_Service>
        <Failure>
        (inFMX | outFMX|comFMX|synchFMX|schedFMX |basExFMX| timeFMX|memFMX)
        </Failure>
        <Reaction> ('avoided'|'detected') </Reaction>
        <IntegrityLevel>' IntegrityLevel </IntegrityLevel>
         (Latency | Error)*
    </Platform_Service>
```

`<Failure>:` a text indicating the type of malfunction anticipated in the platform service
`<Reaction>:` indicates whether the platform is able to avoid the failure when it is detected or to just detect the failure and provide a warning
Latency:
```
    <Latency>
    ('Less than'|'more than') n time_unit ('+ -' n time_unit)* '
    </Latency>
```

Error:
```
        <Error>
         d ('ms')|('us')|('%')
        </Error>
```
;

HealthMonitoring:
```
        <HealthMonitoring> appFailure|failureReaction
                '<IntegrityLevel>' IntegrityLevel </IntegrityLevel>
        </HealthMonitoring>
```

appFailure:
```
        <Failure>'
                <Application>' appFMX '</Application>'
                '<ApplicationResourceName>' resourcename '</ApplicationResourceName>'
    lr=(Latency|Error)*
   '</Failure>'
```

failureReaction:
```
        <Reaction>
                <Failure>
                        (inFMX | outFMX|comFMX|synchFMX|schedFMX |basExFMX| timeFMX|memFMX)
                </Failure>'
                <Action> appFRX </Action>
                <Value> d time_unit (AdditionalConditionOrInfo)* </Value>
        </Reaction>
```

AdditionalConditionOrInfo: a text including A-Z, a-z, 0-9, which cannot start with a number, which provides more information about the time unit.

ServiceDiversity:
```
        <ServiceDiversity>
                <Channel> 'analog'|'digital' </Channel>
                <Operation> 'read'|'write'|'execute' ArrayOfChannelSignalName '</Operation>'
                <CommonCauseType>
                        (inFMX| outFMX|comFMX|synchFMX|schedFMX |basExFMX| timeFMX|memFMX)
                </CommonCauseType>
                <IntegrityLevel>' IntegrityLevel </IntegrityLevel>
        </ServiceDiversity>
```

ArrayOfChannelSignalName: a text including A-Z, a-z, 0-9, which cannot start with a number which indicates the name of channel for the signal

ResourceProtection:
```
<ResourceProtection>'
        <PlatformResource> ('cpu'|'memory'|'platfrom-service')'</PlatformResource>'
        <InterferenceType>
        'interferences'|'temporal interferences'|'spatial interferences'|'behavioral interferences'
        </InterferenceType>'
        <SharingType>'
        ('private/ not shared'|'the resource is spatial partitioned'|'shared, but not at the same
        time'|'shared dynamically')
        </SharingType>
        <IntegrityLevel> IntegrityLevel </IntegrityLevel>
'</ResourceProtection>'
```

comFMX: indicates the possible malfunctions of the communication. It could be any of the following texts: 'Message Corruption', 'Message Insertion', 'Message Loss', 'Incorrect Message Sequence', 'Late Transmission', 'Early Transmission'
;

synchFMX: indicates the possible malfunctions of the synchronization. It could be any of the following texts: 'Mutex Access Commission' | 'Mutex Access Omission' | 'Mutex Release Commission' | 'Mutex Release Omission' | 'Mutex Timeout Failure' | 'Event Signal Commission' | 'Event Signal Omission' | 'Event Timeout Failure'
;
schedFMX: indicates the possible malfunctions of the scheduling It could be any of the following texts:
'Scheduling Jitter Failure', 'Scheduling Deadline Failure', 'Interrupt Handler Latency Failure'

inFMX: Indicates the malfunction of the input signals. It could be able of the following texts:
'Digital Input Omission', 'Digital Input Late Read', 'Digital Input Early Read', 'Digital Input Late Return', 'Digital Input Early Return', 'Digital Input False Positive', 'Digital Input False Negative', 'Analog Input Omission', 'Analog Input Commission', 'Analog Input Late Sampling', 'Analog Input Early Sampling', 'Analog Input Sampling Jitter', 'Analog Input Late Return', 'Analog Input Early Return', 'Analog Input Value Failure'

basExFMX: Indicates the malfunction of the basic execution. It could be any of the following texts
'CPU Failure', 'Main Memory Failure', 'CPU Clock Failure', 'Power Supply Failure'

outFMX: Indicates the malfunction of the output signals. It could be any of the following texts: 'Digital Output Late', 'Digital Output Early', 'Digital Output False Positive', 'Digital Output False Negative', 'Analog Output Late', 'Analog Output Early', 'Analog Output Value Failure'
;
timeFMX: indicates the possible malfunctions in the timing service of the platform. It could be any of the following texts: 'Global Time Failure', 'Relative Time Failure', 'Wait Time Failure'
;
memFMX: indicates the possible malfunctions of the memory. It could be any of the following texts
'Memory Late Read via', 'Memory Read Access Denial via', 'Memory Read Data Failure via', 'Memory Late Write via', 'Memory Write Access Denial via', 'Memory Write Data Failure via'

appFMX: indicates the possible malfunctions of an application. It could be any of the following texts:
'Arrival Rate Failure' | 'Inter Arrival Time Failure' | 'Execution Time Deviation' | 'Logical Sequence Failure' | 'Application Runtime Failure'

appFRX: indicates the possible action of an application when a failure is detected. It could be any of the following texts:
'Restart' | 'Shut Down' | 'Send of the Default Signal' | 'Send of the Default Message' | 'Indication of an Error' | 'Handler Execution'