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	**D6.6 Refined definition of the runtime certificate approach incorporating first concepts regarding the treatment of 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, mailto:werner.weber@infineon.com	
Deliverable responsible	Fraunhofer IESE, Kaiserslautern Daniel Schneider, daniel.schneider@iese.fraunhofer.de	
Version number	v1.0	
Date	30/09/2015	
Status	Final	
Dissemination level	Confidential (CO)	

Authors

Partici-pant no.	Part. short name	Author name	Chapter(s)
01O	IESE	Daniel Schneider	2, 4
01O	IESE	Tiago Amorim	1, 2, 3, 4
01O	IESE	Christoph Dropmann	2, 4
15E	Tecnalia	Alejandra Ruiz	2, 4

Document History

Version	Date	Author name	Reason
v0.1	23/07/2015	Tiago Amorim	Set-up document
v0.1	19/08/2015	Viet Yen Nguyen	Document revision
v1.0	30/09/2015	Tiago Amorim	Review comments included, final version generated
	04/10/2015	Alfred Hoess	Final editing and submission

Publishable Executive Summary

Over the last 20 years, embedded systems have evolved from closed, rather static single-application systems towards open, flexible, multi-application systems of systems. While this is a blessing from an application perspective, it certainly is a curse from a safety engineering perspective as it invalidates the base assumptions of established engineering methodologies. Due to the combinatorial complexity and the amount of uncertainty encountered in the analysis of such systems, we believe that more potent modular safety approaches coupled with adequate runtime checks are required. The contributions of this document are twofold:

- We investigate the possibility of runtime safety support covering both horizontal and vertical dependencies. To this end, we will first elaborate these notions in more detail and provide a brief overview of related research. We will then go into a bit more detail regarding two specific approaches, each of which will be focused on regarding one of these two aspects. We propose an initial concept regarding the integration of these two approaches, leading to the new notion of multidirectional modular conditional certificates. The concept is illustrated based on the industrial use case developed in task 7.3.

- We develop the initial concepts presented in the document *D6.3 Preliminary definition of the runtime certificate approach* regarding security for safety in modular contracts. We answer the initially identified challenges, and propose solutions. We address concepts such as modularization, security properties representation in contracts, contracts counterfeit prevention and required shifts in the current safety certification paradigm.

Table of contents

List of figures

1. Introduction

In recent years we have witnessed two different, very strong trends in the domain of embedded systems: collaboration between systems and more cores per chip.

The trend towards more collaboration has been prevalent for rather closed systems, such as communicating ECUs within a car, for over 20 years now. But roughly since the 2000s, it has been extended towards open systems such as dynamic compositions of different cars, traffic infrastructure, and Internet-based services. New computing paradigms have been coined along the way, most notably pervasive computing, ubiquitous computing, ambient intelligence, and cyber-physical systems. All these notions have in common that different types of systems from different manufacturers are integrated dynamically into so-called systems of systems, which can then render higher-level services based on their collaboration. This clearly bears huge potential for future applications and is bound to make a significant impact on our daily lives. However, from a safety perspective, this trend also brings huge challenges that could well prove to be a show stopper. One key challenge in this regard is the uncertainty regarding dynamic compositions and reconfigurations, which can hardly be foreseen and analyzed at development time already. A corresponding solution idea is to shift parts of the safety certification activities into runtime, where all relevant information can be obtained and uncertainty can be resolved.

The other trend goes is about incrementing the number of CPU cores per chip. This trend has existed for about ten years and leads to higher computing power at lower cost. As a consequence, ECUs can host a higher number of applications at a time or applications that merely require much higher computing power. New kinds of applications are enabled that were previously unfeasible due to high hardware cost or lack of processing power. An example is camera-based recognition applications like those required for autonomous driving. However, from a safety perspective, the rise of multicore architectures has led to the problem of mixed criticalities. Different applications on a multicore processor share the same platform resources, resulting in potentially dangerous interdependencies and interferences. These need to be analyzed thoroughly, measures need to be introduced and their sufficiency needs to be shown. A particular challenge in this regard is cases where applications are developed by different parties or where there is a possibility of dynamic application downloads.

We expect future systems of systems to consist of different collaborating systems (i.e., entities consisting of hard- and software) that might in turn be built upon multicore technology and host several applications. Moreover, dynamic application updates are probably a feature future systems will possess. From a safety engineering perspective, we thus face uncertainties with respect to a system's environment (e.g., other collaborating systems; "horizontal dependencies") as well as regarding different applications of one system (i.e., via common shared resources; "vertical dependencies).

1.1 Objective of this document

Objective of this document is to provide a refined definition of the EMC² runtime certification approach developed in T6.3. The work presented here is built upon on previous work described in the document *D6.3 Preliminary definition of the runtime certificate approach*.

In the context of the EMC² innovation cycle scheme, this deliverable contributes to the MS4 of technology subprojects (cf. Figure 1).

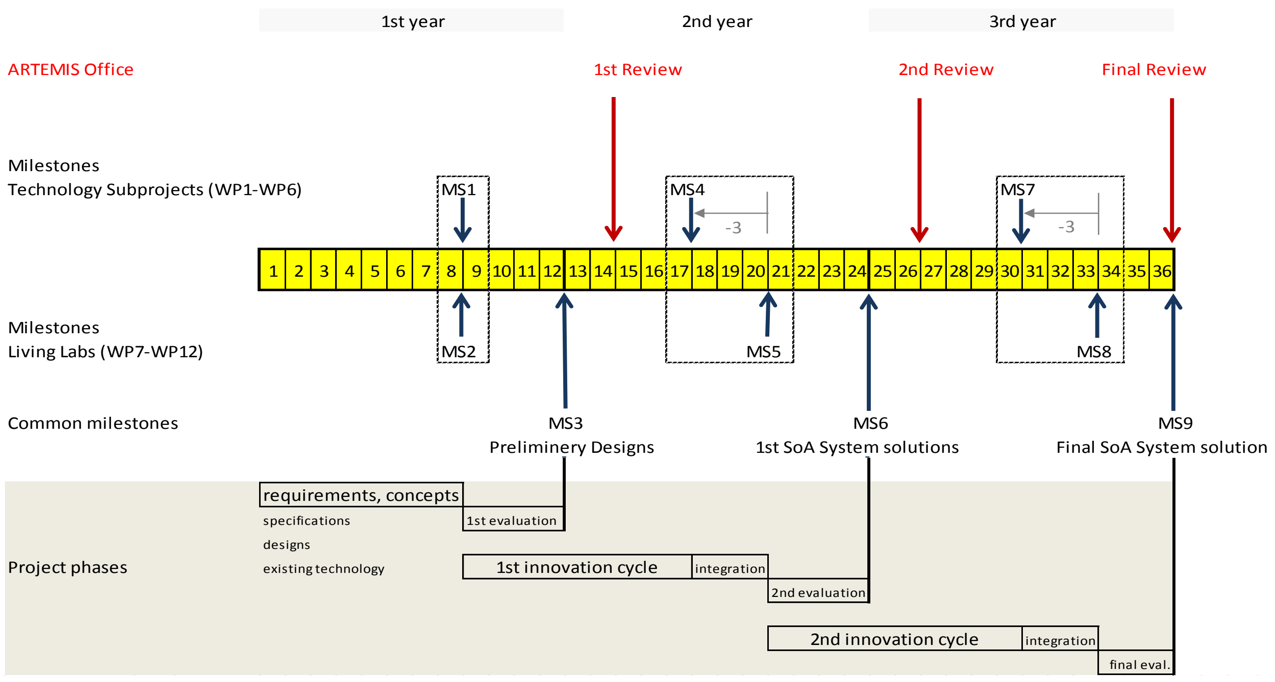

Figure 1 - EMC² milestones plan

This document is organized as follow: Section 2 describes the concept of Multidirectional Modular Conditional Safety Certificates and the concepts associated with it, Section 3 discusses the security for safety in contracts, its implications and application, and Section 4 draws a short conclusion and points next research steps.

1.2 Related deliverables

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

- *D6.3 Preliminary definition of the runtime certificate approach:* This deliverable provides preliminary definition of the runtime certificate approach, main outcome of the Task 6.3 of the EMC² project, and introduces safety assurance concepts for dynamically loaded code. The preliminary definition is based on relevant existing work that is to be combined and tailored to the EMC² context.
- *D7.7 Preliminary UC T7.3 design*: This deliverable provides provide preliminary information on the use case T7.3 "*Design and validation of next generation hybrid powertrain / E-Drive*". The goal of this use case is the tailoring and further enhancement of the EMC² technologies for the design and validation of next generation hybrid powertrains and e-Drives.

2. Multidirectional Modular Conditional Safety Certificates

The Multidirectional Modular Conditional Safety Certificates (M2C2) [1] are an evolution of application runtime safety certification that takes into consideration vertical and horizontal interfaces. The approach is realized through the synergy between VerSaI [2] (in the context of vertical interfaces) and ConSerts [3] (in the context of horizontal interfaces). In the next subsections we will further develop these concepts.

2.1 Related research

In the automotive domain, ISO 26262 [4] is the functional safety standard. Some critical functions are supposed to work independently on a single core with their dedicated resources. When this is not the case, partitioning and shared resource techniques are used. Ruiz [5] state: "The safety assurance argument that these techniques should address is that the presented items of evidence should show sufficiently spatial and temporal independence between each partition." With the introduction of multi-core computers, multiple partitions may run concurrently on a single computing card, all accessing memory or I/O interfaces at the same time and needing to share processing time and resources in a 'safe' way.

Kotaba [6] analyzed the low-level temporal effect from sharing the on-chip resources that impact the execution time. In the "multicore domain, the applications compete for resource access, typically arbitrated in a non-explicit manner by the specific hardware implementation. This causes non-deterministic temporal delays to the execution". They analyzed the effects and suggest mitigation techniques for resources such as system bus, bridges, memory bus and controller, memory (DRAM), shared cache, local cache, Translation Lookaside Buffer, addressable devices, pipeline stages, or logical units.

In SPEEDS [7] a formal meta-modeling language and the syntax of component contracts were developed and implemented. These contracts define the premises and promises of the component regarding its behavior in a specific way as well as an attribute designating its viewpoint. Another project that pursued this idea, CESAR [8], defined the CESAR Meta-Model (CMM), which includes the concept of 'rich' components that can be connected and integrated into hierarchies. Different kinds of rich components are possible depending on the perspective, such as operational actors, functions, logical components, or technical components. The CMM is based on the integration of component-based design with contracts based on input from the SPEEDS project, EAST-ADL2 (traceability, verification and validation) from the ATESST project, and its own CESAR Requirements Management Meta-Model (RMM). All these propose a metamodeling language for components dealing with safety-critical systems.

From another perspective, the FRESCOR project [9] (Framework for Real-time Embedded Systems based on COntRACTS) proposed contract-based resource management in distributed systems. It uses service contracts as a mechanism for dynamically specifying execution requirements. To accept a set of contracts, the system has to check as part of the negotiation whether it has enough resources to guarantee all the minimum requirements specified, while upholding guarantees on all previously accepted contracts negotiated by other application components. If successful, the system reserves enough capacity to guarantee the requested resources and will adapt any spare capacity available to share it among the different contracts that have specified their desire or ability to use additional capacity.

Sljivo [10] introduced the concept of weak/strong assumptions/guarantees when formalizing. They propose contracts in which all properties that an environment shall satisfy are defined separately from those required only in some contexts. This allows their contracts to be used for components and in different contexts.

Ruiz [11] proposes “to formalized contracts through a well-defined and structured contract ‘grammar’ to support how users may systematically assure the safety of their system while integrating components through the definition of a BNF (Backus Normal Form or Backus–Naur Form) grammar”. Ruiz proposes this grammar for design time contracts.

2.2 Use case T7.3 “Design and validation of next generation hybrid powertrain / E-Drive”

In the context of the next generation of hybrid powertrains, our goal is to reduce the efforts and timing due to software updates in order to reduce time to market. These systems need to satisfy the ISO 26262 compliance requirements such as the one mentioned in Part 8, Clause 8.4.5.2 that we shall include “if the change carried out has an impact on safety-related functions, the assessment of functional safety shall be updated.” This means that we need to provide corresponding capabilities for the diagnosis function at runtime, when all systems are known. To achieve this in a safety-critical context, we propose the use of contract-based runtime assurance mechanisms for checking the safety of new updates in a standard and cost-efficient way. It is worth to point that runtime assurance is currently not supported by the standard.

In the figure below, the scenario for our use case is presented. The scenario represents the next-generation hybrid powertrains which is provided by AVL[1] at WP7, task 7.3 and described in the document *D7.7 Preliminary UC T7.3 design*. An electric motor is accessed by an accelerator pedal via a set of software applications. A brief overview of the technical background of the system is given in Figure 2.

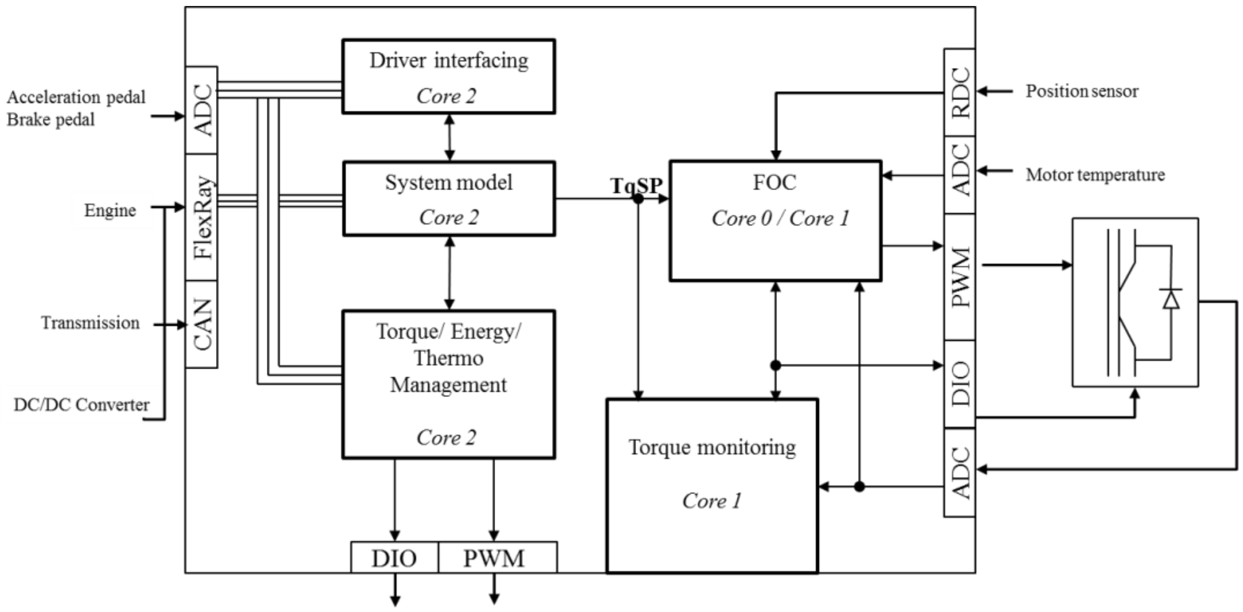

Figure 2 - Powertrain use case context

The System Model located on Core 2 is in charge of receiving data from the accelerator pedal and sends it to Field Oriented Control FOC and Torque Monitoring through the Torque Set Point signal.

The Field Oriented Control (FOC) component allocated in cores 0 and 1controls the speed of the electric motor (E-Motor) based on the Torque Set Point signal received from the system model. In addition, the FOC gets feedback from the electric motor (motor voltage, current, temperature, and rotor position) and communicates with the Torque Monitoring block regarding safety-related information.

[1] http://www.avl.com/

Torque Monitoring performs a plausibility check of the Torque Set Point value based on the signals from the electric motor (voltage and current). It checks if the set value or the FOC violate the system safety by overloading the electric motor. In the case of a potential overload, the monitor sends safety-related information back to the FOC and disables the electric motor.

All these applications need to be safely integrated in order to provide a function that is critical for the vehicle. The objective of this use case is to integrate an update of the torque monitoring functionality into the powertrain system and to ensure that the update is safe and free from interferences with other applications functionalities. An upgraded functionality should undergo an impact analysis in order to be included. A quick and effective way to verify the impact of this upgrade is the use of a runtime contract-based approach to verify the feasibility of this new development and the lack of unwanted interferences between applications through the platform.

2.3 Vertical and Horizontal Dependencies

The concept of vertical and horizontal interfaces was first introduced by Zimmer [12]. 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).

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 independent of the hardware and according to the application's requirements. 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 [13] and the ARINC 653 [14] Integrated Modular Avionics standard. Applications are software components that provide system-level functions to end users or other applications and are not related to services provided by the platform.

The vertical interfaces describe the safety-relevant relations between an application and a platform service. 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 [2] 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. Whether an application demand is satisfied by a service guarantee depends on the consumed service guarantees from other applications. Mitigation and arbitration are needed to realize safe use of the vertical interface and assure compatibility between application demands and platform guarantees.

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 [15] and Tractor Implement Automation [16]. 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. 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 designed level.

2.3.1 Vertical Safety Dependencies

In this subsection, we describe VerSaI, our approach for dealing with vertical safety dependencies. Zimmer [2] proposed a classification of safety-related demand-guarantee dependencies. The dependency classes are platform service failures, health monitoring, service diversity, and resource protection.

Platform service failures 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 is the opposite of platform service failure. 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. An example is the need to develop the analog input channel that is used to read the accelerator pedal in a different way to avoid an analog input value failure as a common-cause failure.

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, e.g., the torque monitoring component must be protected from interferences via the analog-to-digital converter software component that is shared with the FOC.

The demands and guarantees are applied to a confidence attribute. The attribute states the integrity achieved by a guarantee or requested by a demand. Examples are the automotive safety integrity levels or the design assurance levels in avionics.

In the case of an open system like the hybrid powertrain, it is possible 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. Both guarantees and demands can be described using a semi-formal language such as VerSaI (Vertical Safety Interface) [2], which allows automated evaluation if demands and quarantines are compatible.

Platform service failure, health monitoring, and service diversity are vertical dependency classes that arise even for a federated system (with separate platforms for applications). We assume that for systems that integrate applications into a commonly used platform, especially in the context of mixed criticality, the resource protection class is of major interest. For this reason, besides the VerSaI guarantees and demands, we focus on an automated platform service interference analysis in combination with a protection assignment. This allows safe and automated integration of additional applications during a product's lifetime.

2.3.2 **Horizontal Safety Dependencies**

ConSerts is our starting point as a solution for horizontal dependencies. It stands for Conditional Safety Certificates. ConSerts was initially presented in [17] [18] [3]. It 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. This approach was also presented in the previous document, *D6.3 Preliminary definition of the runtime certificate approach*.

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. This certificate evaluation can be realized off-board (performed by an extra system) and/or on-board (the systems support runtime evaluation). 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 re-evaluation and the determination of a new one. 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).

2.4 Approach description

The vertical interface has direct influence on the performance of the application since it is the platform that provides the physical resources to run the application. If the required application demands are not completely fulfilled by the platform guarantees, the application cannot deliver its full capabilities with the designed confidence. Thus, this has a direct impact on the horizontal services guarantees provided by the application to other applications. In other words, the horizontal guarantees of the application are influenced by the fulfillment of its vertical demands.

The mediation of horizontal and vertical interfaces becomes highly relevant if different applications are integrated in the same platform, consequently influencing the horizontal guarantees provided by the application services. This occurs in the context of mixed criticality and dynamic updates where re-evaluation is required to assess whether the overall demands, both vertical and horizontal, are properly satisfied.

To address the aforementioned issues, we introduce Multidirectional Modular Conditional Certificates (M2C2), a runtime certification approach that addresses both vertical and horizontal interfaces. The approach is realized through the synergy between VerSaI (in the context of vertical interfaces) and ConSerts (in the context of horizontal interfaces).

In ConSerts, a service guarantee can be correlated to demands and their fulfillment by other application services' guarantees. In M2C2, the services are additionally influenced by the guarantees of the platform. This relation is illustrated in Figure 3. During M2C2 contract resolution, the vertical application demands shall be fulfilled by platform guarantees before resolving the horizontal relations. If some of the vertical demands are not fulfilled, some of the application-service guarantees at the horizontal interfaces might not get validated.

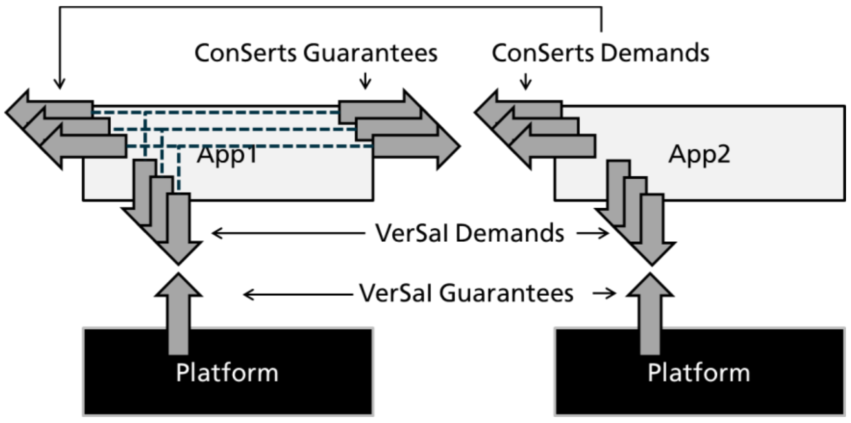

Figure 3 - Relations between ConSerts Guarantees and VerSaI Demands

Resolving vertical dependencies on a single application running on a platform is very straightforward. The platform guarantees and application demands are compared and either match or do not match. However, if a platform hosts more than one application (several single core platforms being combined in one multi-core), the applications might influence each other's behavior, even if there are no horizontal interfaces between them. Interferences as described in section 2.3 can occur. To guarantee segregation between the integrated applications, the platform needs to allocate its resources consequently considering the tradeoff between the demands of each application and the required rendered services/safety levels.

In a situation where the vertical application demands cannot be satisfied for a given configuration of applications, a new configuration needs to be identified. This results in an iterative approach in which the platform resources are re-allocated to critical applications while non-critical or less critical ones get to share what is left. This can even lead to the removal of existing applications.

Besides the capability to address dependencies between applications as well as between applications and their platform, the M2C2 approach presented in this section exhibits two important synergies.

First, the combination of the horizontal and vertical contracts in one consideration allows additional flexibility. For instance, if the platform's guarantees are not sufficient, due to an accidental or intended alteration such as a partial platform service breakdown or an application download, the information can be propagated to the horizontal interface. Subsequently, the alteration propagates through the guarantee-demand relationships of affected applications, potentially resulting in an alteration of the "top-level" guarantees of the current overall system configuration. This kind of propagation could either be fostered by means of the Boolean logic employed by ConSerts or, alternatively, there might be a dedicated centralized mediator component as part of a corresponding runtime framework. This mediator would then be responsible for monitoring the established contracts and, in case of deviations, would calculate a new safe set of vertical and horizontal contracts based on the current conditions.

Second, as part of the vertical interface we want to detect and address all potential interferences between applications. Besides interferences, there are intended interactions between applications as well. The interference analysis can point out such intended interactions in addition and forward them as a warning to the horizontal interface. Hence, unconsidered interactions between applications can be revealed. For instance, in the use case from section 2.2, the DIO (Digital Input/Output) is used by the torque monitoring and the FOC applications. This could either be intentional or an error in the integration design, e.g., if the FOC as a less safety-critical component could enable the motor when the torque monitor application tries to disable the motor.

2.5 Applying the M2C2 framework to the use case

To provide a better understanding of the aforementioned approach, we will describe a practical application with the help of the aforementioned use case scenario. The use case describes a processor with three cores running applications that control an electric powertrain. These applications are distributed among the cores and share several resources such as communications channels, which implies possible influences between those applications. Besides, some applications must interact with each other on the horizontal level and require (from its peers) services with a minimal integrity level. The M2C2 framework can be used to address these issues.

We assume that the vertical application demands are specified by the application developers and the platform guarantees are specified by the platform developer. An example is that the FOC and the Torque Monitoring applications specify demands regarding the analog-to-digital converter (ADC) service, which does, for instance, comply with the AUTOSAR specification. Our vertical approach then consists of two steps. First, health monitoring and service diversity will be performed to examine if all application demands can be fulfilled by the ADC service guarantees. Second, the resource protection dependencies will be evaluated. Consequently, it will be evaluated if the shared use of the service is without interferences between the applications.

After the vertical dependencies have been assessed positively, the next step is the resolution of the horizontal dependencies. The System Model (SM), Torque Monitoring (TM), and Field Oriented Control (FOC) are the applications that participate in the presented use case and their relations are depicted with arrows in Figure 4. Their certificates are defined with ConSerts, which is described using EBNF grammar in order to facilitate runtime resolution. An example of a service guarantee is SystemModel.SMTorque(1): ASIL = c, Late{10ms;}.ASIL.d. This guarantee is bound (e.g., via the Boolean logic of ConSerts; cf. [3]) to a demand directed at the platform. The service in the example is implemented in ASIL c and guarantees that the application passes the signal in less than 10ms with a confidence level of ASIL d. The demand representation has a similar format. The TM and FOC demands need to be lower or equivalent to the service guarantee provided by the SM. The same happens between TM and FOC and vice-versa (since they have a closed-loop relation and are provider and consumer of each other’s services at the same time). Once all demands are fulfilled by the respective corresponding guarantees, the overall system can be considered as safe. Due to space and scope limitations, the syntax and semantics of the contracts will not be further detailed, for more information; the reader shall refer to [3] and [2].

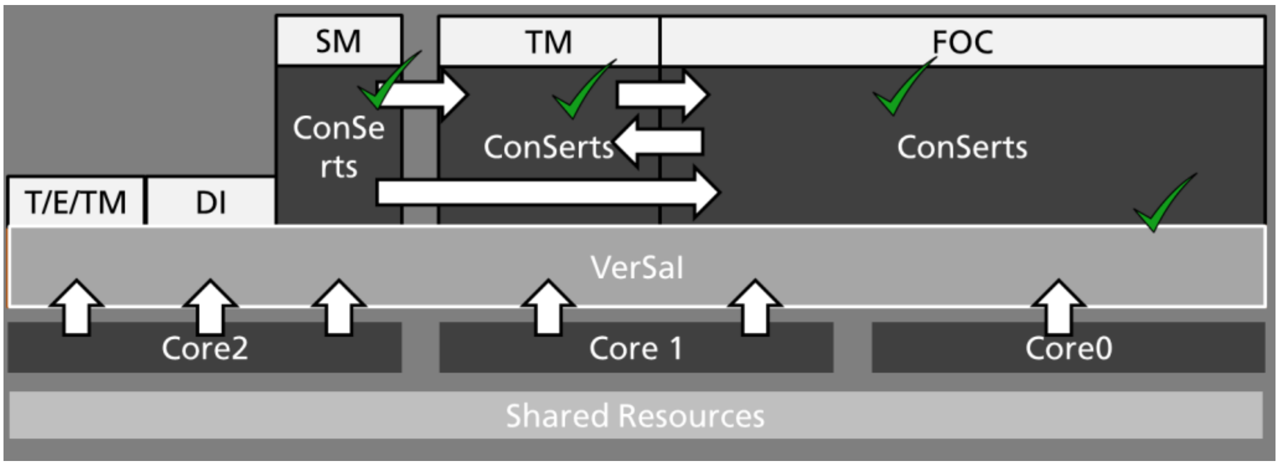

Figure 4 - M2C2 certificate verification applied to the use case (The direction of the arrows represents a guarantee (from) being provided to a demand (to).)

An overview of the M2C2 certificate assessment is illustrated in Figure 4. Note that although the applications Torque/Energy/Thermo Management (T/E/TM) and Driver Interfacing (DI) do not participate in the horizontal interface assessment, they are considered in the vertical interface step. In case of a change in the initial application configuration, i.e. if new applications are added or applications are replaced by others with more functionality, a new assessment needs to take place to guarantee that the change will not jeopardize the intended integrity level.

M2C2 addresses the adaptive behavior of systems at application level, taking into consideration the resources provided by the platform (platform guarantees) and the demands of the applications. M2C2 allows that different combinations of applications are tested in the platform until the most important (and safety critical) have enough guarantees to execute their functions, even if this would mean removing not so critical applications from the platform.

3. Security for safety

Cyber-Physical Systems (CPS) offer tremendous promise. Yet their breakthrough is stifled by deeply-rooted challenges to assuring their combined safety and security. All relevant safety standards assume that a system’s usage context is completely known and understood at development time. This assumption is no longer true for Cyber-Physical Systems (CPS). Their ability to dynamically integrate with third-party systems and to adapt themselves to changing environments as evolving systems of systems (CPSoS) is a headache for safety engineers in terms of greater unknowns and uncertainties. Also, a whole new dimension of security concerns arises as CPS are becoming increasingly open, meaning that their security vulnerabilities could be faults leading to life-endangering safety hazards.

Despite this, there is no established safety and security co-engineering methodologies (or even standardization). In fact, their respective research communities have traditionally evolved in a disjointed fashion owing to their different roots: namely embedded systems and information systems. With CPSoS, this separation can no longer be upheld.

As described in the document *D6.3 Preliminary definition of the runtime certificate approach*, the open nature of cyber physical systems bring security problems that is already a reality in the automotive domain [19], with some car manufacturers doing recalls to fix security breaches in their systems [20]. In the rest of this chapter we will present our current findings regarding security for safety and how to operationalize it through runtime contracts.

3.1 Mismatching points between Safety and Security

Safety is commonly defined as the absence of unacceptable risks. These risks range from random hardware failures to systematic failures introduced during development. 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, each in its own way. The following issues, in particular, are intrinsically in conflict [21]:

1. **Assumed User Intention:** Safety deals with natural errors and mishaps, while security deals with malice from people (i.e., attacks). Thus, safety is able to include the user in its protection concept, whereas security distrusts the user.
2. **Quantifying Risks:** Safety practices utilize hazard probability when defining the acceptable risk and required safety integrity level of a system function. In security, measuring the likelihood of an attack attempt on a system in a meaningful way is impossible. Error and mishaps can, to a certain degree, be quantified statistically, whereas it is unfeasible to estimate the occurrence of an attack. An attacker's motivation may change over time.
3. **Protection Effort:** Safety is always non-negotiable. Once an unacceptable risk has been identified, it must be reduced to an acceptable level, and the reduction must be made evident based on a comprehensive and convincing argument. Security, in contrast, is traditionally a trade-off decision. Specifically in the information systems domain, where a security issue is associated with a monetary loss (in some form), the decision about how much effort to invest into protection is largely a business decision.
4. **Temporal Protection Aspects:** 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 depreciates as a result of increases in computational power, development of attack techniques, and detection of vulnerabilities. This is such a huge issue that a system might require a security update the day after it goes into production. Consequently, effort for ensuring safety is mainly concentrated in the design and development phases. In the case of security, the effort is divided among design, development, and operations and maintenance, the latter requiring higher effort.
5. **COTS:** Safety-critical systems benefit from COTS. In such widely used and tested components, design flaws and failure probabilities are known. In terms of security, COTS can be detrimental since the design of these components is usually publicly available and found vulnerabilities can be exploited wherever the component is used.

3.1.1 Challenges tailoring security aspects in safety contracts

In the previous document, *D6.3 Preliminary definition of the runtime certificate approach*, we presented some challenges that needed to be addressed in order to tailor safety contract with security aspects. In the following paragraphs we present how we decided to tackle those challenges:

- **How security guarantees and demands should be modelled and how this should be represented in contracts? –** Security supports and protects the safety functions, which means that the security counter measures are also responsible for keeping the system safe, like safety functions do. In this perspective, security countermeasures belong to the set of evidences that assures the safety of a system and, therefore, should be treated as such. However, the contracts should take into account the updates required to keep the countermeasures efficient.

- **How security can be modularized?** – The modules with open interfaces, w.r.t modules that can become a gateway for a security attack, should be able to withstand those attacks and not propagate the harm to other modules. In this perspective, modularization of security is not a concern since this issue must be dealt and contained within the module. However, the scrutiny of the respective security countermeasures is an important factor and reflects on the safety integrity level of the component and associated services provided.
- **How to always keep a system secure since this is not a static property?** – In a cyber-physical context, keeping a system secure means also to keep it safe. In this matter, safety contracts will not be complete if they do not assure that the security mechanisms are up-to-date i.e. all known vulnerabilities are covered. This requires having part of the certificate being dynamic and changing according to the latest security patch applied.

These ideas are further developed in the following section, in which we better define the role that security plays when defining safety contracts.

3.2 Security in contracts

In this section, we expand the answers the presented in the previous sections and present some initial solutions for the identified challenges.

3.2.1 Openness categories

Systems can have variable degree of openness, regarding how it connects to other systems and which kind of networks/devices can be used. We could identify the following openness categories:

- **Completely open systems:** These systems are connected to the Internet all the time. At the same time that is possible to attack such a system from anywhere on the planet, they also are able to keep the latest security updates, even being subscribed to a "push" security update schema, in which the updates are sent to the system by the software developer. Despite Zero-day-exploits, these systems can be considered the safest.
- **Partially open systems:** These systems have open non-physical interfaces (wireless) but are not connected to the Internet. They are less exposed than completely open systems but their security updates aren't applied as frequently as the previous category. Attacking those systems requires some proximity or the use of other systems that can be used as bridge.
- **Partially closed systems:** These systems do not have open wireless interfaces, but they have open physical interfaces that can be exploited by an experienced attacker. Examples are USB ports that allow memory-sticks to be read, or media centers that can run malicious software burnt on media discs. This kind of attacks requires more creativity and experience but they are still feasible. A classic example is the Stuxnet virus that can use USB memory-sticks to infect networks.
- **Strongly closed systems:** These systems connect to others through physical proprietary interfaces such as ISOBUS in the agricultural domain. They could be hacked only through other systems.

3.2.2 Modularization

A module with open interfaces (interfaces with vulnerabilities that could be exploited for a cyberattack) need to be able to withstand these attacks and not propagate any harm to other modules. In the context of closed modules, it would not make sense to have modules propagating safety hazards due to design flaws, why it should be any different when considering security? The conditional certificates states that the guarantees provided by a module are reliable and enough to trust the services provided by that module. The same line of thought should be applied to security induced errors. The module with open interfaces should be able to not propagate any misbehavior originated from a security attack, which could develop into a safety hazards. In case a module with open interfaces cannot guarantee its own security (due to poor security countermeasures or lack of updates) that module cannot be considered safe and its guarantees should be defined accordingly, otherwise it will mislead the other modules and make the overall system unsafe.

As discussed before, safety is a static property while security depreciates over time. This characteristic of safety is also reflected in the contracts, once they are defined, there is no need to change them. For a safety contract that also considers security we need to care about the freshness of the security counter measures. To this matter, we envision the contract having two parts, a safety part that is static and stays the same through all product life span and a changeable part, responsible to represent when the latest security update was applied to the system.

Let’s analyze a safety contract and try to identify intersection points between security characteristics. In ConSerts, the safety contracts are represented using EBNF-style grammar and can be evaluated using Boolean logic [3]. Bellow you can appreciate an example of contract:

setPTOE: AgPL = c, Late{10s, Standstill}.AgPL = d

Where the “setPTOE” stands for the service type being provided, “AgPL = c” is the Service-Level Integrity Level (IL) and it implies that all safety properties of the service are at least guaranteed with that IL. The last part, “Late{10s, Standstill}.AgPL = d”, refines specific safety properties, in this case the service will not be later than 10 seconds while in Standstill with confidence of AgPL d. Further information about building contracts using ConSerts can be found in [3].

Like safety, security also provides services, but, usually, they are not the main functions of the system. Authentication mechanisms, for example, exist to provide security for transactions to be performed, by themselves they are useless. Describing security services in contracts, the same way it is done for safety ones, is not necessary.

The Service-Level IL of a safety contract describes the scrutiny applied for the development of the safety critical parts. The assurance of the security countermeasures are related to the safety functions it supports. Security counter measure implementation should follow the same criteria of the aimed criticality level of the function it aims to protect [22]. The assumption is, if there is need for security countermeasures they should be implemented to protect the safety critical parts and the countermeasures implementation will be compatible with the associated safety criticality.

If a countermeasure protects more than one safety function, it needs to be implemented in such way that fulfills the integrity level of the most critical function.

We need to transmit the information that the security countermeasures are still good, which means saying that the latest security patch was installed. The security part of the contract shall inform other modules when the last update was performed and the version of the latest update. In this sense, every new update requires a new contract. We envision that the following information should be in the contract in the security side:

Field	Description
Latest patch version	Describes the version of the latest patch installed in the system
Latest patch date	Describes when the latest patch was installed.
Patch validity date	For systems that have recurrent patching routine, this field tells when the current patch expires and a new patch needs to be installed.
System manufacturer	Tells who the manufacturer of the system is. This allows other system to know which Public key to use in order to check the contract authenticity.

Table 1 - Initial definition of security contracts

3.2.3 Asymmetric cryptography

The contracts should be security protected against counterfeits. Counterfeit contracts can pose great danger to other system since we cannot trust that the other system will truly behave as stated. Our envisioned solution is to protect and certify contracts using Asymmetric cryptography, orchestrated by the software company developer and overseen by a safety certification agency.

In order to prevent forgery or tampering, the contracts should be digitally signed using asymmetric cryptography. With this algorithm, the validity of the certificate can be checked using a public key provided by the manufacturing company. Meanwhile, every new update should be reported to the certification agency which will keep the safety evidences for the new certificate.

Asymmetric cryptography is used in open networked environments in order to prevent security problems. It is able to preserve confidentiality, integrity of the communication and authenticity of the sender, also ensuring non-repudiation or non-denial of the sending. The technique works with two keys, a private key used for encryption and a public key, that can be distributed and is used only for decryption. The public key is widely distributed while the private key is kept in secrecy. The public key allows other parties to verify if a certificate is valid but it does not allows someone to forge a certificate, since for this the private key is required and it is known only to its proprietor. Both keys are related mathematically, however calculating the private key from the public key is unfeasible due to the parameters chosen.

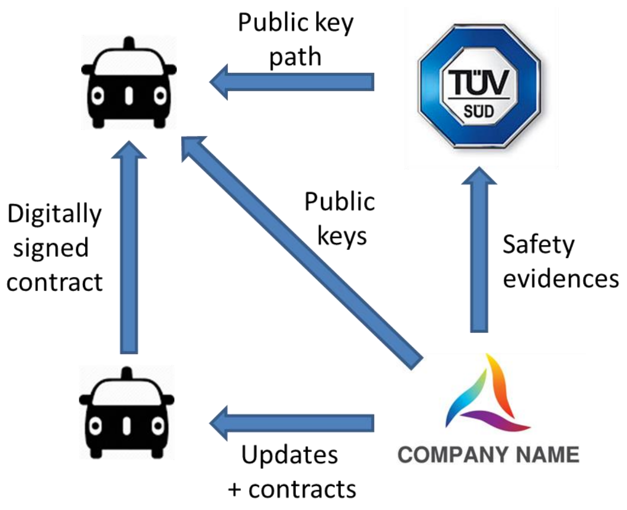

Figure 5 - Contract asymmetric cryptography orchestration

When two systems need to check the validity of their contracts, the respective public key is required. The key must be fetched from the contract manufacturer servers. The contract has the manufacturer name which is used to fetch the path to the manufacturer from the safety certification agency. We considered having the manufacturer path directly in the contract, however counterfeit contracts could lead the systems to access malicious links and we wanted to avoid that. This whole process is depicted in Figure 5.

The certificates should be digitally signed, sent and installed together with security updates. The authenticity of the certificate can be verified later using the system manufacturer’s public key. This can prove that the sender (software manufacturer) had access to the private key and guarantees the origin of the certificate.

It is possible, with asymmetric cryptography to verify the validity of certificates offline, by collecting public keys in advance. This allows keys to be stored (e.g. in memory sticks) and uploaded in machines that do not have (constant) access to internet connection. Besides, the retrieved public keys can be stored locally and used to check if contracts of other system are authentic. Also a schema like Cookies used in web browsers could make the process more efficient. A good application for this are commuters using autonomous cars, where people are very likely to meet the same people while going to work due to routine habits. Cookies allow the cars to know they already “know” each other. Bellow we list the identified advantages and disadvantages of using such approach:

Advantages

- **Preventing counterfeit certificates**: If we cannot guarantee that the certificates are really the ones provided by the manufacturing organizations we can run into safety problems. Hacked systems could use fake certificates in order to be able to interact with other safety critical systems. Rooted systems might also impose hazards since they are modified versions of the systems created by producers and usually are not self-updating as the original software.

- **"Patch validity date" forces frequent updates**: It is possible to define an expiration date for a certificate. This forces the system to constantly update the certificate, even if new security updates are not available yet.
- **Offline usage**: The public keys required to check the authenticity of a contract can be downloaded in advance and the process can take place without connection to the certificate provider. This is useful for closed systems that connect to open systems or with partially open systems (systems that have open interfaces but are not connected to the Internet).

Disadvantages

- **Revoking certificates might be troublesome**: If the private key gets disclosed, revoking the digitally signed certificates is a solution and would be possible, however this is not a simple process. A public key revoking list has to be made available for systems that want to check some certificate validity. Besides, systems without access to the revoke list might run into counterfeit certificates without a way to figure this out.
- **Computational expensive**: Compared to symmetric key algorithms, the asymmetric key algorithm is more computationally intensive, this, added to the fact that embedded systems suffer from resource restriction can hinder the usage of this algorithm in some systems.

3.2.4 **Certification paradigm shift**

The current established safety certification paradigm (in some domains) has the following workflow: first, the software is designed and developed while safety engineering takes place and safety evidences are collected. Then, these evidences are sent to a certification agency, responsible for assessing the overall safety of the system. If the process goes well, the system is allowed to be used by civilians. This whole process can take weeks.

From a security perspective, attack techniques evolve quite fast. For this reason, several updates to fix exploitable vulnerabilities are required during the whole system lifecycle. The frequency of these updates can vary from weekly (e.g. Patch Tuesday [22]) to daily. To illustrate such issue we present below some data from the National Vulnerability Database [23] with the number of vulnerabilities found between 2010 and 2014:

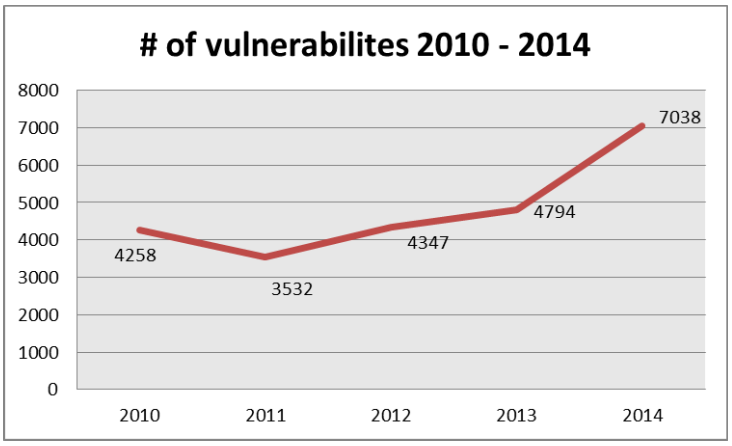

Figure 6 - Number of vulnerabilities per year between 2010 and 2014 [23]

In current safety certification state-of-the-practice, if a system is changed after certification process is completed, the system is required to go through a re-certification process. Besides being very time consuming, re-certification is also costly. Re-certificating a system weekly after a security update is released is not very efficient, when following the current paradigm. At the moment, we envision two possible solutions for this problem:

- **Identify changeable parts of the system:** Security counter-measures will need to be updated. By not doing it, the system will become vulnerable, therefore unsafe. The idea is to identify the parts that are more likely to be updated beforehand and making sure that this does not affect the safety critical parts. The certification of the safety critical part stays the same as long as this part remains untouched. The implementation needs to take care so that there is low coupling between parts that need change and parts that should stay the same.
- **Release update before evidences are assessed:** Although only the certification agencies can assess if a system is safe enough, the manufacturers have guides and know how to produce safe systems and the required evidence for its certification. This process needs to be changed and the system developer should gather evidence that the system is safe and send to certification agency at the same time the update is released. This way, if the system causes any harm, the provided evidences will prove that the safety problems were not caused due to bad design of the system. The evidences provided can be used legally.

Security updates should be released as soon as possible. The longer it takes to have a system patched by a security update, the more likely is to an attacker to exploit its vulnerability. System updates and safety certification are incompatible nowadays. Any change in the system requires re-certification, and this needs to be carried out before the system is released to the general public. Therefore, a shift in the safety certification paradigm is required.

4. Conclusions

In this document, we presented an approach aimed at supporting systems dependability through modularization of safety assessment. The proposed Multidirectional Modular Conditional Certificates (M2C2) framework is a novel runtime certification approach because it addresses both vertical and horizontal safety interfaces. Vertical interfaces describe dependencies between applications and a platform and horizontal interfaces describe dependencies on the system level between applications or systems of systems. Besides a detailed discussion of horizontal and vertical aspects, we presented a first idea for merging ConSerts, an approach for the horizontal safety interface, and VerSaI, an approach for the vertical safety interface.

The resulting M2C2 framework allows negotiating whether a system consisting of applications integrated on common platforms is safe or not at runtime. In addition, the combination of the two perspectives allows mitigating interferences between applications through the platforms in such a way that safety can be ensured in combination with maximal application service availability instead of a failsafe. The applicability of the framework ranges from applications running within a processor core (like the presented use case) to coarse-grained systems of systems (such as cyber-physical systems).

We also extended our initial ideas for integrating security properties in safety contracts. We evolved the initial set of identified challenges and further developed the usage of contracts. We sketched an initial contract architecture schema, trying to pinpoint advantages and disadvantages of each idea. The work presented here paves the way for the final form of the safety-security certificates and its implementation in industry.

4.1 Next Steps

As future work, we plan to refine the concepts regarding the mitigation and the implementation of the presented approach to evaluate and demonstrate that M2C2 is feasible. Strategies can be explored in case of the run time check failing completely. Some initial ideas are self-adaptation of the applications or graceful degradation.

Regarding the safety-security certificates, the presented ideas need to be better refined and validated with practitioners; this shall be performed in the context of EMC² and the Industrial Living Labs. With this action, we expect to identify issues in order to have an approach better aligned to the state-of-the-practice and suited for the current needs of Industry.

5. References

[1] 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*, Delft, 2015.

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

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

[4] International Organization for Standardization, "ISO26262 Road vehicles – Functional safety," 2011.

[5] A. Ruiz, I. Habli and H. Espinoza, "Towards a Case-Based Reasoning Approach for Safety Assurance Reuse," in *Computer Safety, Reliability, and Security, Lecture Notes in Computer Science*, 2012.

[6] O. Kotoba, J. Nowotschy, M. Paulitschy, S. M. Pettersz and H. Theiling, "Multicore In Real-Time

Systems – Temporal Isolation Challenges Due To Shared Resources," in *WICERT workshop*, 2013.

[7] SPEEDS Project, "D.2.5.4 Contract Specification Language (CSL); Deliverable; Rev. 1.0.1," 2008.

[8] CESAR, "D_SP1_R3.3_a_M3 Meta-Model Concepts for RTP V," CESAR Project.

[9] Frescor project, „Framework for Real-time Embedded Systems based on COntRACTS," [Online]. Available: http://www.frescor.org. [Zugriff am 04 05 2015].

[10] I. Sljivo, J. Carlson, B. Gallina and H. Hansson, "Fostering Reuse within Safety-critical Component-based Systems through Fine-grained Contracts," in *International Workshop on Critical Software Component Reusability*, 2013.

[11] H. E. T. K. Alejandra Ruiz, „Adequacy of contract grammars for component certification," in *Safecomp*, Toulouse, France, 2013.

[12] 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.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] A. Greenberg, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," WIRED, 21 07 2015. [Online]. Available: http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. [Accessed 04 08 2015].

[20] "Nach Auto-Hack: Fiat Chrysler ruft 1,4 Millionen SUV zurück," Spiegel Online, 24 07 2015. [Online]. Available: http://www.spiegel.de/auto/aktuell/nach-hackerangriff-fiat-chrysler-ruft-suvs-von-jeep-zurueck-a-1045294.html. [Accessed 04 08 2015].

[21] 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.

[22] C. Schmittner und Z. Ma, „Towards a Framework for Alignment between Automotive Safety and Security Standards," in *DECSoS 2015: EWICS/ERCIM/ARTEMIS Dependable Cyber-Physical Systems and Systems-of Systems Workshop*, Delft, 2015.

[23] C. Budd, "Ten Years of Patch Tuesdays: Why It's Time to Move On," GeekWire, 31 10 2013. [Online]. Available: http://www.geekwire.com/2013/ten-years-patch-tuesdays-time-move/. [Accessed 04 08 2015].

[24] "National Vulnerability Database," National Institute of Standards and Technology, [Online]. Available: https://nvd.nist.gov/. [Accessed 04 08 2015].